University of Cyprus

Deliverable 6:

Measurements for the Evaluation of IP Architectures and Protocols of Concern.

ENDIKTIS

Table of Contents

1 Introduction

The existing Internet architecture is based on the “best effort” model for delivering packets across the Internet. The current architecture delivers a packet at its best possible (best-effort) but doesn’t guarantee when it will be delivered. The demands of the users have changed dramatically since the creation of IP, where it was mostly used for email and ftp. Another new application is the WWW that has been widely used worldwide. WWW has created a new friendly interface for the user, and stimulated further demands from the network.

The existing architecture of IP is inadequate to handle new applications. Time critical applications such as video, audio and several others have created an even greater demand on the Internet. Lately, several new protocols and architectures are proposed to enable basic quality of service provision in Internet.

In this deliverable we conduct extensive experiments – in the form of simulations and pilot networks – in order to use the measurements taken from the results to evaluate IP architectures and protocols of concern. We specifically concentrate on the differentiated services for the provision of quality of service in IP networks by examining existing active queue management schemes that provide congestion control.

2 Development of Pilot networks

2.1 Differentiated Services Pilot Network

A differentiated services (Diff-Serv) pilot network in Linux environment is implemented, and the performance of various network functions are investigated that may provide differentiated quality of service (QoS). These functions include various queuing disciplines for providing adequate congestion control. Through the pilot network we aim to investigate different ways to implement differentiated networks and present recommendations for different network traffic and conditions.

2.1.1 Introduction to Linux Networking Services

Linux is an open source operating system, which is freely available to the public. Linux had gained popularity all over the world but mostly in the academic environment. Most of the testbeds are released in Linux or in Unix environment first. Linux offers a rich set of Traffic Control (TC) functions for networking.

Lists of possible network traffic control functions include:

- Throttle bandwidth for certain computers
- Throttle bandwidth to certain computers
- Fairness for bandwidth sharing
- Multiplex several servers as one, load balancing, or enchanted availability
- Restrict access to your computers
- Limit access of your users to other hosts
- Do routing based on user id, MAC address, source IP address, port type of service, time of day or content.

The Linux kernel offers support for Diff-Serv and QoS. Before we get into the details of traffic control configuration of Linux we have to understand how the TC works under Linux. In order to transmit data into the network we have to setup the network card, using appropriate driver software.

Two functions of the driver software are:

- The Linux Networking Code can request the network driver to send a packet on the physical network.
- The network driver can deliver packets that it has received on the physical network to the Linux Networking Code. The current architecture sends the data from the application to the networking driver, see Figure 1.

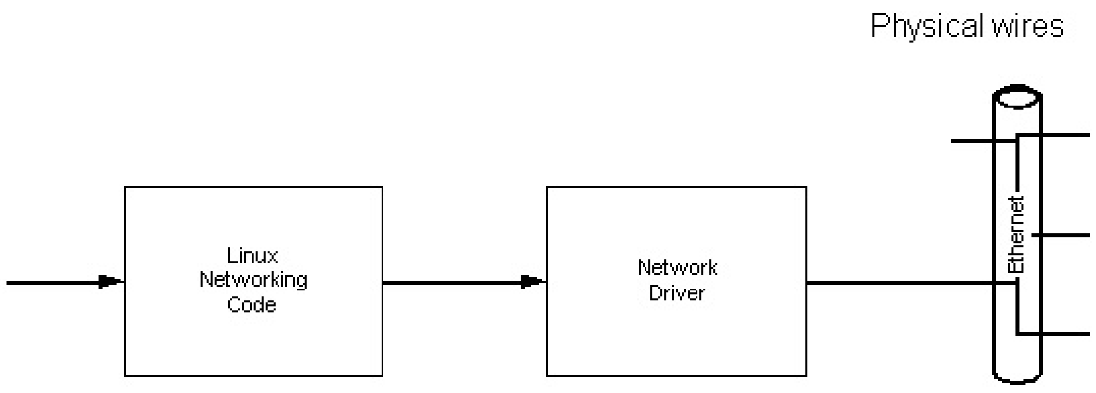


Figure 1: Default Setup of a Linux

Figure 2 shows an extra function, the Traffic Control function, included in the LINUX implementation. With the traffic control in between the Linux Networking Code and the Network driver, packets can be manipulated in several ways.

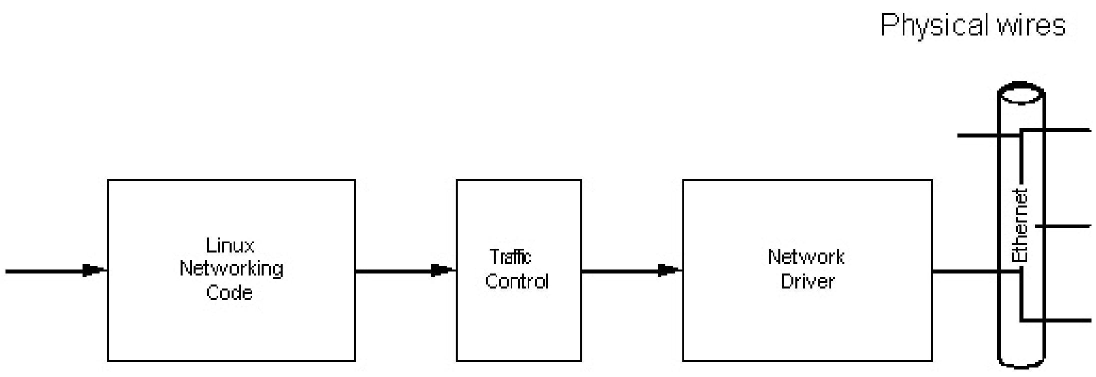


Figure 2: Linux Setup with Traffic Control

Figure 3 shows the block diagram of the kernel processes, the packets received from the network and how new data is generated to be sent on the network [1].

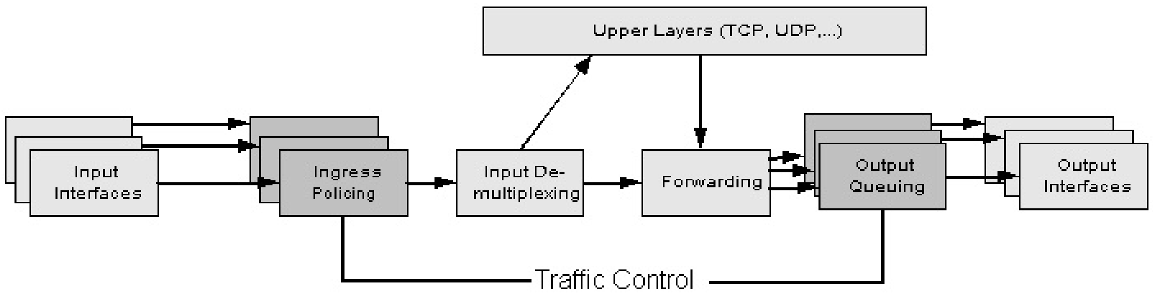


Figure 3: Processing of network data

The Input interface is responsible for passing packets to the Ingress Policing module. Packets could be policed at the Input interfaces. The Ingress policing modules are responsible for discarding traffic in the event that packets are arriving too fast. Then the packets are either forwarded on different interface, in case the machine is acting as a router, or are passed to the higher layers for further processing. The forwarding module is responsible for the selection of the output interface, the selection of the next hop, encapsulation etc. Then the packet is queued at the output interface. The traffic control can drop packets based on several parameters that can be selected by the user.

The major conceptual components of the traffic control of Linux code are:

- Queuing disciplines
- Classes (with queuing disciplines)
- Filters
- Policing (and related concepts)

2.1.1.1 Queuing Discipline

Every network device has its own queuing discipline that controls how the packets are enqueued. There are several queuing disciplines such as pFIFO, bFIFO, SFQ, RED and so on. Figure 4 shows a queuing discipline.

Queuing discipline

Figure 4: A simple queuing discipline without classes.

Figure 5 shows a queuing discipline that uses filters to prioritize packets into to different classes. More than one filter can be mapped to a class [2].

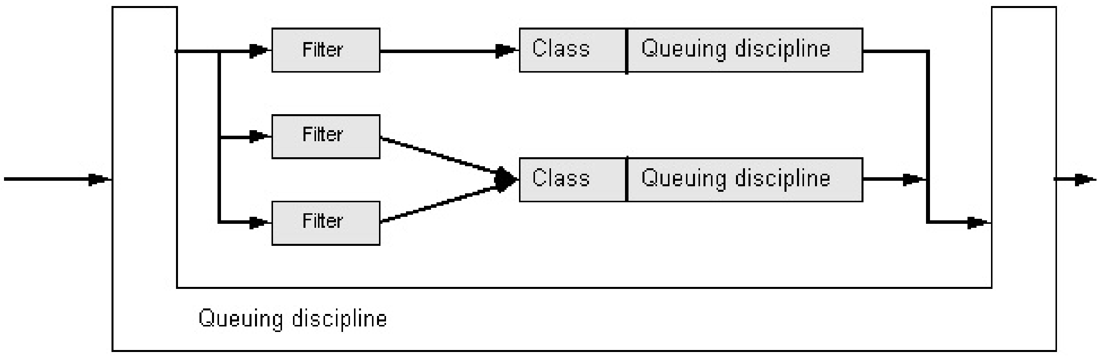


Figure 5:A simple queuing discipline with multiple classes.

Classes use another queuing discipline to store their data; such queuing disciplines can be pFIFO, bFIFO, RED, SFQ and etc. Figure 6 shows this scenario.

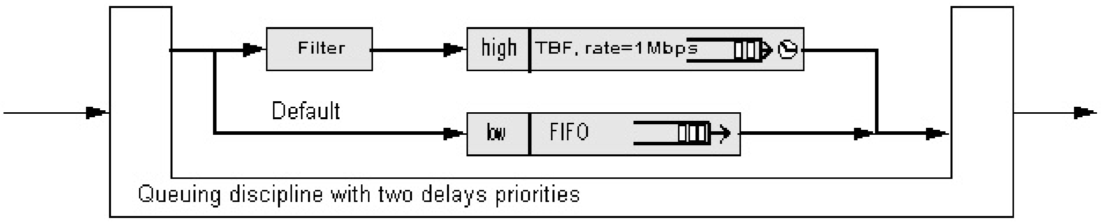


Figure 6: Combination of priority. TBF and FIFO queuing disciplines.

The queuing discipline implements a two-delay priority. The packets are filtered and classified into these two classes. The first queue is a token bucket filter, which is the high priority class. The TBF is served with 1Mbps rate and has a higher priority than the FIFO. All the other packets are classified into the lower priority queue, which is served with a queuing discipline FIFO.

Each queuing discipline is identified by unsigned 32 bit numbers, u32. The identification number of the queuing discipline is split into two parts, the major number and the minor number. The major number and the minor number are 16 bit each. The notation is major:minor, where a minor number is always zero for the queuing disciplines, see Figure 7. At the network device eth0, there must be only one queuing discipline, which the major number of the queuing discipline must be unique. In case the user doesn’t define the major number of the queuing discipline the system assigns one automatically.

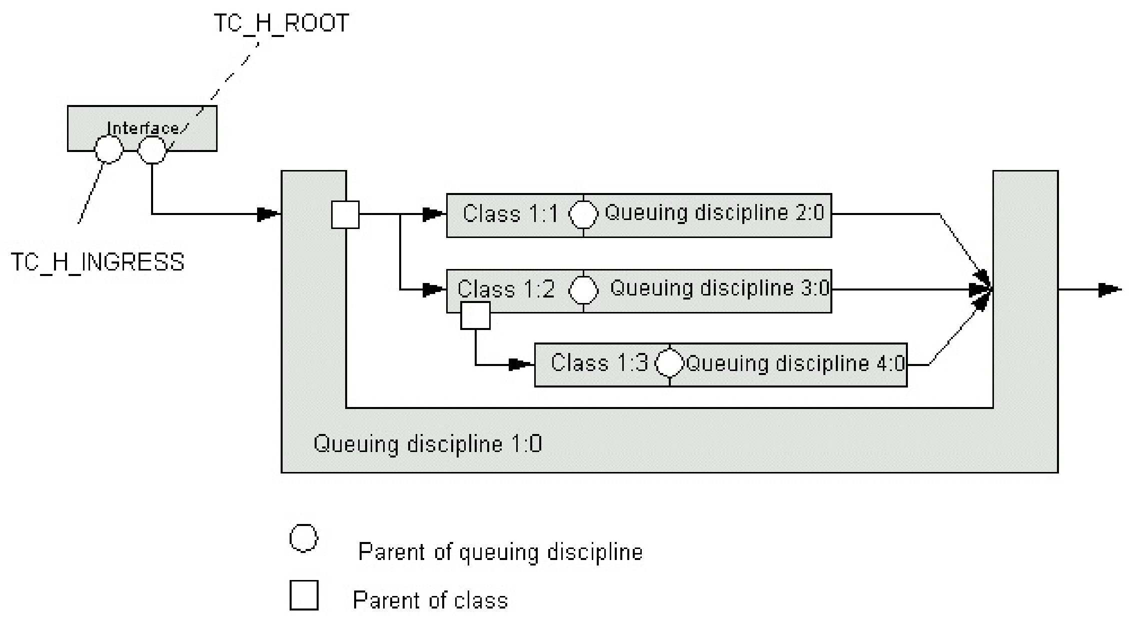


Figure 7: Addressing for queuing disciplines and classes.

Each queuing discipline has a set of certain functions that uses to control its operations. Such functions are enqueue, dequeue, requeue, drop, init, reset, destroy, change, and dump. More detail description of these functions can be found at [3]. There are some statistics that are maintained by each queuing discipline. The minimum sets of statistics that are maintained are the following:

- The current queue length
- The cumulative number of bytes enqueued
- The cumulative number of packets dequeued
- The cumulative number of packets dropped

2.1.1.2 Classes

There are two ways that you can identify a class: by the class ID, and the internal ID. The class ID is been assigned by the user, and the internal ID by the queuing discipline. The internal ID must be unique with a given queuing discipline. The data type of the Class ID is u32 and the internal ID is unsigned long. The kernel is accessing the class by its internal ID.

Queuing disciplines with classes provide a set of functions to manipulate classes. A list of these functions is graft, leaf, get, put, change, delete, walk, tcf_chain, bind_tcf, unbind_tcf and dump_class [3].

2.1.1.3 Filters

The incoming packets have to be assigned into the various classes. This is done using filters. Queuing disciplines are responsible for this task and with the usage of filters can assign incoming packets into classes. This takes place during the enqueue operation. The filters are organized based on the queuing discipline or class. All filters are stored in a filter list. This list is organized either by the queuing discipline or by class. It's also ordered based on the priority, in ascending order. The structure of the filters can been seen at Figure 8.

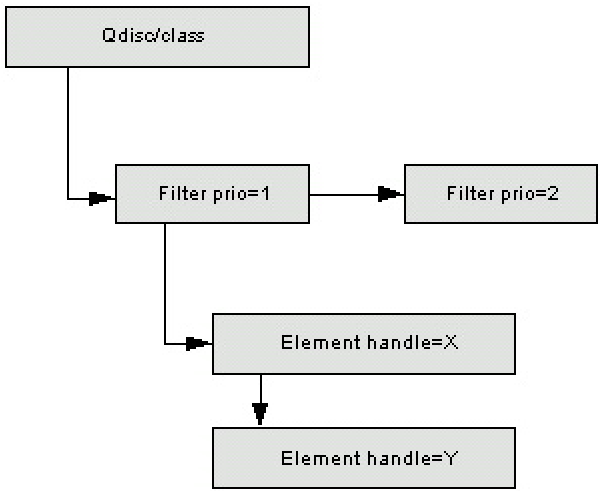


Figure 8: Structure of filters

Like classes, filters have internal IDs that are used to be reference for some internal tasks. Figure 8 shows the handles and the internal ID that are used for internal purposes. These handles are 32-bit and the internal IDs are unsigned long type. The order of which the filters and their elements are examined to get a match for the incoming IP is shown at Figure 9.

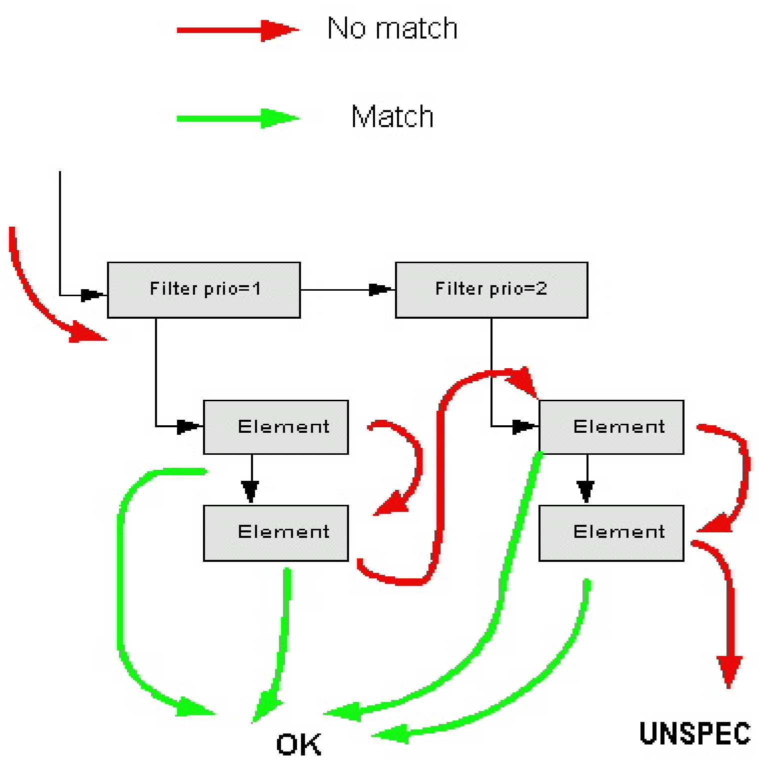


Figure 9: Looking for Filter Matching

There are several functions that can be used in order to control the filters. A list of these functions is classify, init, destroy, get, put, change, delete, walk, and dump. For more information regarding these functions can be obtained from [1,3]. Filters are broken down to generic and specific filters. Generic filters can use one instance per queuing discipline that can classify packets for all classes. The cls_fw, cls_route, and cls_tcindex are generic filters. Specific filters use one or more instances of the filter or its internal element per class. The cls_rsvp and cls_u32 are specific filters.

2.1.1.4 Policing

In order to make sure that the traffic doesn't violate a certain limitation, we use policing. The policing is broken down to five policing mechanisms: policing decisions by filters, policing at ingress, refusal to enqueue packets, dropping packets from an inner queuing discipline and dropping a packet when enqeueing.

2.1.1.5 Classifiers under Linux

Some of the classifiers that are used by the **tc** program are the following:

- ***fw***

Bases the decision on how the firewall has marked the packet.

- ***f32***

  Bases the decision on fields within packet (source-destination address, etc)

- ***route***

  Bases the decision on which route the packet will be routed.

- ***rsvp, rsvp6***

  Bases the decision on the target (destination, protocol) and optionally the source as well.

The classifiers that we have listed above accept several parameters that some of them are common. A list of these parameters follows:

- ***protocol***

  The classifer defines the protocol that will accept. Required IP only.

- ***parent***

  The handle this classifier is to be attached to. This handle must be an already existing class. Required.

- ***prio***

  Defines the priority of this classifier.

- ***handle***

  This handle means different things to different filters.

2.1.2 Evaluating Differentiated Services on Linux

2.1.2.1 Topology Under Study

Figure 10 shows the network setup that we have implemented. The clients are connected on a 100Mbps switch and the outgoing interface of the router goes on a 10Mbps Hub.

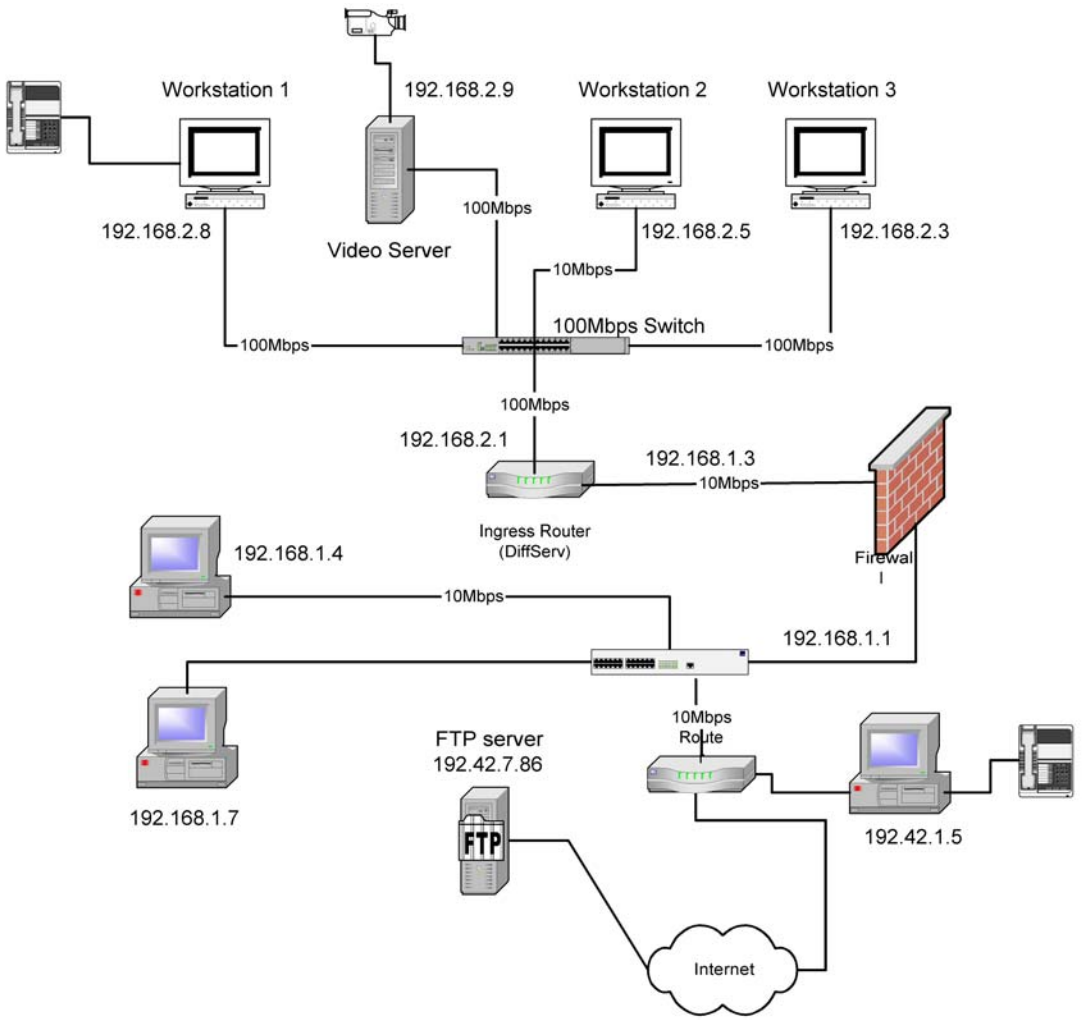


Figure 10: Network Topology for Diff-Serv Architecture

In order to obtain some statistics we have used various tools such as IPERF, TCPDUMP, TCPTRACE and XPLOT. IPERF generates UDP and TCP data traffic. It has the ability to transmit the data at specific port, or at specific bit rate, or a certain amount of data. IPERF runs under Linux and Windows. Another tool that we have used is the TCPDUMP. TCPDUMP captures the traffic at the Ethernet card. And last, we have used the TCPTRACE tool. TCPTRACE analyses the data that are generated from the TCPDUMP. TCPTRACE generates some files that can be used by the XPLOT tool to generate graphs.

2.1.2.2 *Evaluating Linux Implementation for Diff-Serv*

This section focuses on the evaluation of the Diff-Serv architecture under Linux. Extensive experiments have been conducted, with several Scenarios being considered, which aim to show the behavior of the Linux implementation of Differentiated Services under various queuing disciplines, topologies, various parameters etc. First we give an overview of the scenarios we have investigated (see Table 1-4).

SCENARIO 1						
	Scheduler	Class Priority	Class Weight	Filter Priority	Queue Types	Traffic Type
Test 1						
	CBQ	5	500K	1	pFifo	UDP
		5	500K	1	pFifo	UDP
Test 2						
	CBQ	5	500K	100	pFifo	UDP
		5	500K	1	pFifo	UDP
Test 3						
	CBQ	1	500K	1	pFifo	UDP
		5	500K	1	pFifo	UDP
Test 4						
	CBQ	4	500K	1	pFifo	UDP
		5	500K	1	pFifo	UDP
Test 5						
	CBQ	5	800K	1	pFifo	UDP
		5	200K	1	pFifo	UDP
Test 6						
	CBQ	5	500K	1	pFifo	UDP
		5	500K	1	pFifo	TCP
Test 7						
	CBQ	5	500K	1	pFifo	TCP
		5	500K	1	pFifo	TCP

Table 1

Scenario 2						
Test 1						
	Scheduler	Class Priority	Class Weight	Filter Priority	Queue Types	Traffic Type
	CBQ	5	200K	2	TBF	UDP
		5	800K	1	pFifo	UDP
Test 2						
	CBQ	5	200K	2	TBF	UDP
		5	800K	1	pFifo	TCP

Table 2

Scenario 3						
	Scheduler	Class Priority	Class Weight	Filter Priority	Queue Types	Traffic Type
Test 1						
	PQ	PQ1	200K	5	pFifo	UDP
		PQ2	800K	5	pFifo	UDP
Test 2						
	PQ	PQ2	200K	5	pFifo	UDP
		PQ1	800K	5	pFifo	UDP

Table 3

Scenario 4						
	Scheduler	Class Priority	Class Weight	Filter Priority	Queue Types	Traffic Type
Test 1						
	CBQ	5	300K	2	pFifo	UDP
		5	700K	1	RED	UDP
Test 2						
	CBQ	5	300K	2	pFifo	TCP
		5	700K	1	RED	TCP
Test 3						
	CBQ	5	300K	2	pFifo	UDP
		5	700K	1	RED	TCP
Test 4(TCP Window size 64K)						
	CBQ	5	500K	2	pFifo	TCP
		5	500K	1	RED	TCP
Test 5 (TCP Window size 128K)						
	CBQ	5	500K	2	pFifo	TCP
		5	500K	1	RED	TCP

Table 4

2.1.2.2.1 Scenario 1 (pFIFO, pFIFO)

The network topology is set as shown in Figure 11 for all tests conducted regarding Scenario 1.

Test 1

In this scenario we are using two pFifo queues, see Figure 11. The source 192.168.2.5 generates traffic for the receiver 192.168.1.4. The flow 1 represents the traffic that travels from 192.168.2.5 through the upper pFifo and to the 192.168.1.4. The flow 2 represents the traffic that starts from 192.168.2.3 and traverse through the lower pFifo towards to 192.168.1.7. From Table 5, we can see that both of the flows have the same priority and the same weight. Both of the sources transmit the same amount of data at the same bit rate. In all the tests, we have weighted the super class of the CBQ at 1Mbit.

Flow 1 and flow 2 are 5Mbps each. Table 5 shows the transferred data and the bit rates that we have been transmitting from the sources.

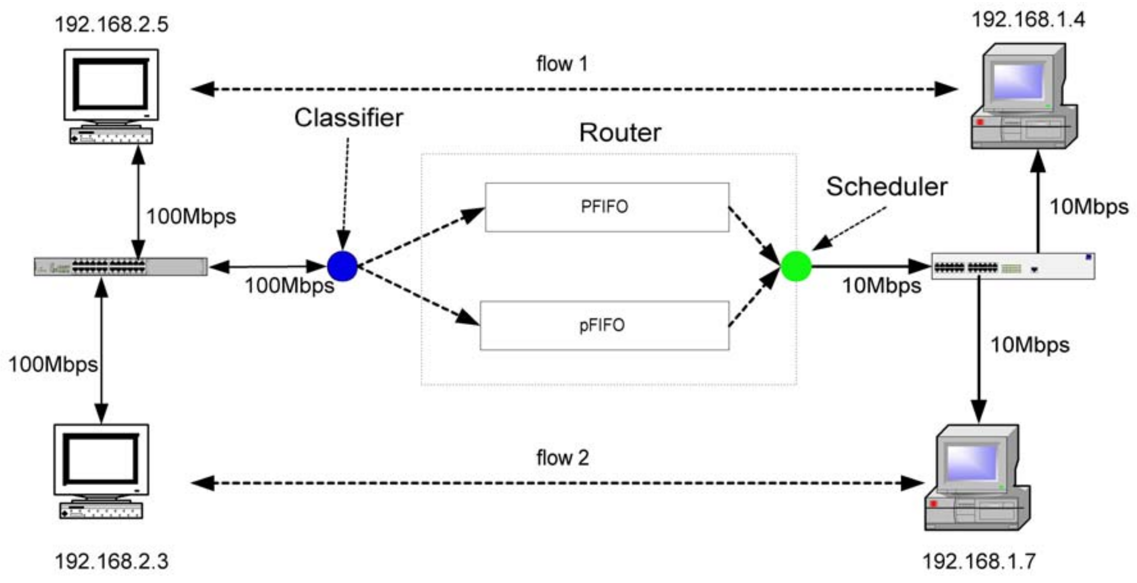


Figure 11: Block Diagram of pFifo queues

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	500K	1	5Mbps
192.168.2.3	U32	CBQ	5	500K	1	5Mbps

Table 5

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	pFifo	50	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	pFifo	50	UDP	14267	22.9

Table 6

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	4.5Mbps	13.1Mbytes	9.294	4926	14267	23.1
192.168.1.7	4.6Mbps	13.1Mbytes	1.421	4918	14267	23

Table 7

The results from this test can be seen in Table 6 and Table 7. The results appear reasonable since the incoming rate at the receivers is below 5Mbps. This test shows fairness over the UDP flows.

Test 2

In this test, we would like to investigate the effect of the filter priorities. Table 8 shows the setting of the filter priorities.

The results are similar to the test 1; see Table 9 and Table 10. In this test, observe that the filter priorities do not play a critical role on the bandwidth allocation. We have to keep in mind that the filter priorities are for the classification of the data into the queues. Of course, we cannot assume the same if we were sending TCP and UDP traffic at the same time. An important observation is calculated jitter of the two flows. IPERF, the tool that we are using, uses the formula 1, to compute the jitter.

$$E\{(W_i)-E[W_i])]\} \qquad (1)$$

where, Wi is a random delay that rises out of the buffering within network

After analyzing the formula we can notice that the results are correct. Keep in mind that was impossible to synchronize the two sources to start transmitting data at the same time. By knowing this, one of the two sources can take the advantage of not giving very accurate results. We noticed the following behavior; at the starting time the source that started transmitting first had a different jitter than the other. At the period that both of the sources were transmitting we noticed the same jitter at both ends, and towards the end we noticed again different jitter time since one of them had finished transmitting.

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	500K	100	5Mbps
192.168.2.3	U32	CBQ	5	500K	1	5Mbps

Table 8

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	pFifo	50	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	PFifo	50	UDP	14267	22.9

Table 9

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	4.5Mbps	13.0Mbytes	8.294	5029	14267	23.1
192.168.1.7	4.6Mbps	13.3Mbytes	1.345	4913	14267	23

Table 10

Test 3

In this test, we set the class priorities. Flow 1 has a higher priority over flow 2, see Table 11. All the other parameters remain the same as were in test 1.

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	1	500K	1	5Mbps
192.168.2.3	U32	CBQ	5	500K	1	5Mbps

Table 11

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	pFifo	50	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	pFifo	50	UDP	14267	22.9

Table 12

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	7.0Mbps	20.0Mbytes	0.708	0	14267	22.9
192.168.1.7	2.1Mbps	6.2Mbyte	11.459	9876	14267	23.1

Table 13

Table 12 and Table 13 show that we can differentiate flows with higher priorities. The results are remarkable since we got rates up to 7Mbps on the 192.168.1.4. The lower the number is set at the class priority, the higher the priority it has over the other flow. When a class has a higher priority over the other one, the scheduler has to execute the packets in that class and then move to the next one.

Test 4

In test 4 we investigate the sensitivity of the priority level (see Table 14). For instance, what is the relation between two classes that have priority 1 and 5 and 4 and 5.

The results (see Table 15-16) of this test show that there is not much of a differentiation among the distances of the priorities. As far we got a difference among the priorities is good enough in order to give a higher priority to the queue.

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	4	500K	1	5Mbps
192.168.2.3	U32	CBQ	5	500K	1	5Mbps

Table 14

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	pFifo	50	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	pFifo	50	UDP	14267	22.9

Table 15

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	7.0Mbps	19.9Mbytes	0.683	47	14267	22.9
192.168.1.7	2.2Mbps	6.2Mbyte	12.690	9838	14267	23.1

Table 16

Test 5

Here we investigate the weight behavior of a class. In order to accomplish this, we change the weights of the classes to be non-proportional to their bandwidths.

Table 17 shows the parameters of the routers. The weights are 800K for flow 1 and 200K for flow 2.

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	800K	1	5Mbps
192.168.2.3	U32	CBQ	5	200K	1	5Mbps

Table 17

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	pFifo	50	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	PFifo	50	UDP	14267	22.9

Table 18

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	6.6Mbps	19.0Mbytes	0.683	691	14267	22.9
192.168.1.7	2.5Mbps	7.2Mbyte	8.838	9154	14267	23.1

Table 19

Table 18 and Table 19 show the results of this test. The results confirm the expectation, since the incoming traffic at the receivers is adjusted based on the weight value. This behavior is expected since the weights take place when the class priorities are the same.

Test 6

In this test, we change the traffic type that we are generating at the sources (see Table 20). The source 192.168.2.5 generates UDP traffic and the 192.168.2.3 generates TCP traffic.

It's obvious (see Table 21-22) that UDP is getting most of the bandwidth at 6.9Mbps and the TCP is getting 3.0Mbps. This behavior is as expected, since the UDP traffic rate is not controlled. In contrast TCP is flow controlled, using a variant of the Jacobson algorithm. Observe that the uncontrolled UDP behavior has a substantial number of losses (recall no flow control, no sensing for retransmission of lost packets).

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	500K	1	5Mbps
192.168.2.3	U32	CBQ	5	500K	2	5Mbps

Table 20

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	PFifo	50	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	3.8	1514	PFifo	50	TCP	14267	41.7

Table 21

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	6.9Mbps	19.9Mbytes	0.736	100	14267	22.9
192.168.1.7	3.8Mbps	20Mbytes	----------	0		41.7

Table 22

Test 7

Test 7 sets both sources to use TCP (see Table 23).

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	500K	1	5Mbps
192.168.2.3	U32	CBQ	5	500K	2	5Mbps

Table 23

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	3.5	1514	pFifo	50	TCP		46.5
192.168.2.3	192.168.1.7	20	3.4	1514	PFifo	50	TCP		47.5

Table 24

Destination Results			
Destination IP	Incoming Traffic	Data Transferred	Time (s) Duration
192.168.1.4	3.5Mbps	20Mbytes	46.5
192.168.1.7	3.4Mbps	20Mbytes	47.5

Table 25

We can see from Table 24 and Table 25 that using TCP in both flows we get a fair treatment over the packets. We have to note that in TCP we mostly get zero packets losses, since the TCP window size is not big enough to exceed the bandwidth rate. At later a stage, we will show the effect of increasing the TCP window size.

2.1.2.2.2 Scenario 2 (TBF, pFIFO)

In this scenario, we change queuing disciplines. Here we use a Token Bucket Filter, TBF, at the upper queue and pFIFO at the lower queue. Also, we have allocated the bandwidth differently from the previous scenario. Here we give 2Mbps to the upper queue of the TBF queue and 8Mbps to the pFifo. Even though we have allocated 2Mbps to the TBF class, the TBF have been configured to limit the traffic at 1.5Mbps.

The network topology is set as shown in Figure 12 for all tests conducted regarding Scenario 2.

Test 1

In this test we have set up both of the classes with the same priority. The weights have been set based on the bandwidth allocation. The flow 1 gets 1.5Mbps and flow 2 gets 8Mbps. Both of the sources generate UDP traffic at 7Mbps.

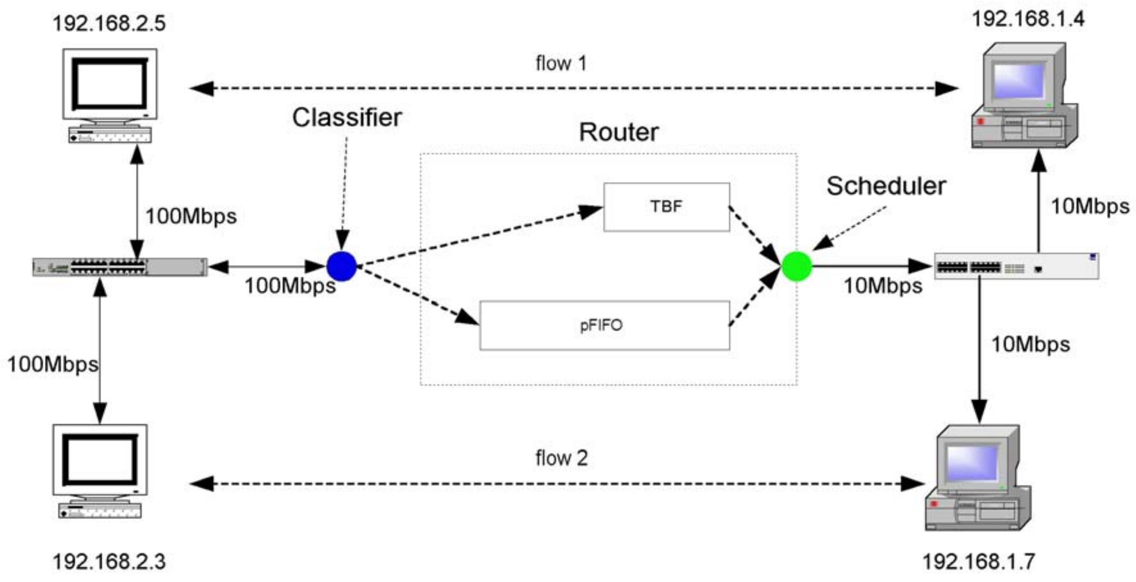


Figure 12: Block Diagram of TBF. and pFifo queues

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	200K	2	2Mbps
192.168.2.3	U32	CBQ	5	800K	1	8Mbps

Table 26

The parameters of the TBF are the following:

- Rate 1.5Mbps
- Burst 1.5KByte
- Limit 1.5Kbytes

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	TBF		UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	pFifo	50	UDP	14267	22.9

Table 27

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	1.1Mbps	3.2Mbytes	15.313	11979	14267	23.1
192.168.1.7	7.0Mbps	20Mbyte	0.268	0	14267	22.9

Table 28

From the results (see Table 27-28) we can see that the incoming traffic at the receivers is almost what we have expected, even though this behavior leads to the presence of losses at the UDP traffic.

Test 2

In this test we transmit UDP traffic through flow 1 and TCP through flow 2 (see Table 29).

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	200K	2	2Mbps
192.168.2.3	U32	CBQ	5	800K	1	8Mbps

Table 29

The parameters of the TBF are the following:

- Rate 1.5Mbps
- Burst 1.5KByte
- Limit 1.5Kbytes

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	TBF	50	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	pFifo	50	TCP	14267	28.0

Table 30

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	1 Mbps	2.8Mbytes	21.664	12255	14267	23.1
192.168.1.7	5.7Mbps	20Mbyte				28.1

Table 31

Table 30 and Table 31 show something very interesting. Using, TBF we can limit the UDP traffic from stealing traffic from other classes, even though there are some losses at the UDP traffic.

2.1.2.2.3 Scenario 3 (Priority Queues)

In this scenario we evaluate the Priority Queues. The main focus here is to see the behavior of the PQ.

Test 1

The PQ discipline is executing first the queue with the highest priority and then the rest. In our test we have assign flow 1 to be classified at Priority 1, which has the highest priority and flow 2 on the lower priority queue (see Table 32).

Router Setup Parameters				
Source IP	Classifier	Scheduler	Class Priority	Filter Priority
192.168.2.5	U32	PQ	PQ1	5
192.168.2.3	U32	PQ	PQ2	5

Table 32

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	pFIFO	50	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	pFIFO	50	UDP	14267	22.9

Table 33

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	7.0Mbps	20Mbytes	0.759	0	14267	22.9
192.168.1.7	2.2Mbps	6.2Mbyte	8.322	9829	14267	23.1

Table 34

As it is expected (see Table 33-34) flow 1 gets most of the bandwidth. This means that it has priority over the others. It’s obvious that flow 1 gets more priority than flow 2.

Test 2

In order to prove the consistency of this test we have reversed the priorities of the flows (see Table 35-37). Flow 1 is assigned to the priority queue 2. This means that it has lower priority over the others.

Router Setup Parameters				
Source IP	Classifier	Scheduler	Class Priority	Filter Priority
192.168.2.5	U32	PQ	PQ2	5
192.168.2.3	U32	PQ	PQ1	5

Table 35

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	pFIFO	50	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	pFIFO	50	UDP	14267	22.9

Table 36

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	2.2Mbps	6.2Mbytes	3.371	9839	14267	23.0
192.168.1.7	7.0Mbps	20Mbyte	0.735	0	14267	22.9

Table 37

2.1.2.2.4 Scenario 4 (pFifo, RED)

Diff-Serv architecture has been focusing on various Per Hop Behavior groups. One of the most important one is the Expedited Forwarding. In this scenario, we have implemented an EF PHB, and we have analyzed it to a certain extent. The network topology is set as shown in Figure 13 for all tests conducted regarding Scenario 4.

Test 1

In test 1 we have two queues; a pFifo (upper queue) and a RED (lower queue). In this test the priorities of the classes and the filters are set the same. The weights of the classes are proportional to the bandwidth of the classes. Flow 1 has 3Mbps and flow 2 has 7Mbps. Both sources transmit UDP traffic at 7Mbps (see Table 38).

The parameters for the RED queue are the following:

- Limit 60KB
- Maximum 45KB
- Minimum 15KB
- Probability 0.1

From the results (see Table 39-40) we get 2.7Mbps and 6.4Mbps for flow 1 and flow 2 respectively. Flow 1 is been limited at 3Mbps and flow 2 at 7Mbps with a considerable amount of packet losses.

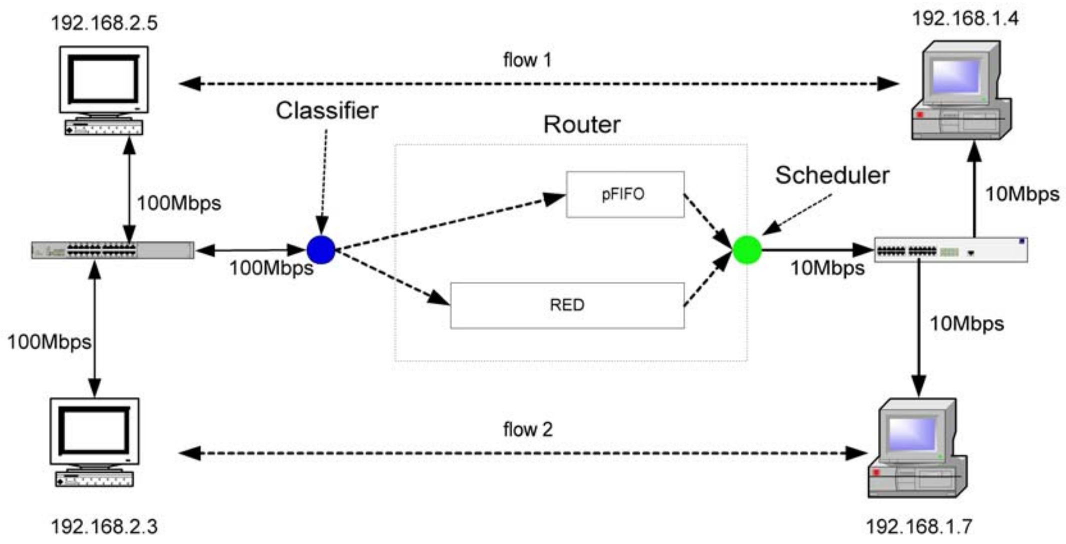


Figure 13: Block Diagram of EF PHB

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	300K	1	3Mbps
192.168.2.3	U32	CBQ	5	700K	2	7Mbps

Table 38

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	7	1514	pfifo	10	UDP	14267	22.9
192.168.2.3	192.168.1.7	20	7	1514	RED		UDP	14267	22.9

Table 39

Destination Results						
Destination IP	Incoming Traffic	Data Transferred	Jitter Delay (ms)	Packet Loss	Total Packets	Time (s) Duration
192.168.1.4	2.7Mbps	7.9Mbytes	0.708	8659	14267	23.1
192.168.1.7	6.4Mbps	18.2Mbyte	0.596	1253	14267	22.9

Table 40

Test 2

In this test we change the traffic type of the sources. Both sources transmit TCP traffic (see Table 41).

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	300K	1	3Mbps
192.168.2.3	U32	CBQ	5	700K	2	7Mbps

Table 41

Source Results								
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Time (s) Duration
192.168.2.5	192.168.1.4	20	3.2	1514	pfifo	10	TCP	50.6
192.168.2.3	192.168.1.7	20	3.5	1514	RED		TCP	45.7

Table 42

Table 42 shows the results. At the receivers we get 3.5Mbps and 3.2 Mbps for flow 1 and flow 2 as the sending rate of the sources is below the available bandwidth.

Test 3

In this test we repeat test 2 with focus on the queue behaviors. The setup parameters have been changed. We are allocating 5Mbps per flow, and we set the same class priorities. Both of the sources are transmitting TCP traffic. The TCP window size is 64K bytes at the sources and the receivers.

The weights of the classes are the same 500K each class (see Table 43).

The parameters for the RED queue are the following:

- Limit 60KB
- Maximum 45KB
- Minimum 15KB
- Probability 0.1

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets Transmitted	Time (s) Duration
192.168.2.5	192.168.1.4	20	3.6	1514	pfifo	10	TCP		44.2
192.168.2.3	192.168.1.7	20	3.4	1514	RED		TCP		47.0

Table 43

In this test we observe some losses over flow 2, (107 lost packets). These packets are caused by RED since it is dropping packets based on the probability that we have assigned to the queue length. Figure 14 and Figure 15 show the outstanding packets of both sources.

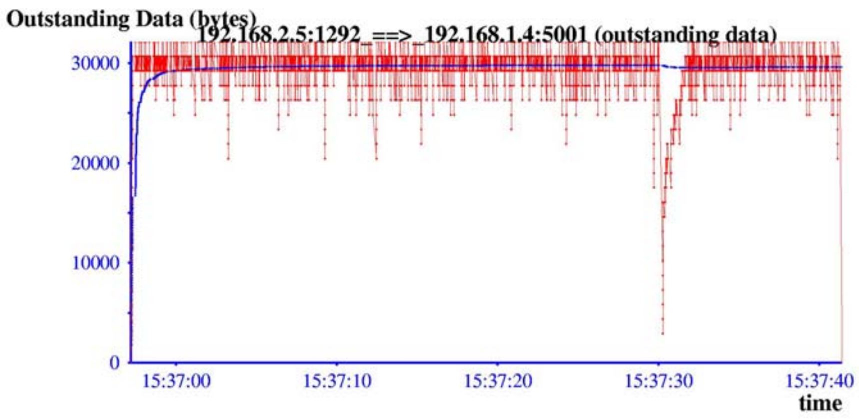


Figure 14: Outstanding Data

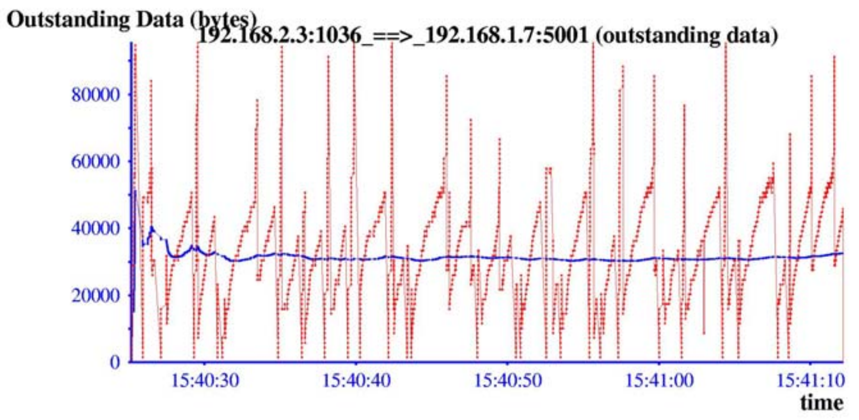


Figure 15: Outstanding Data

Figure 16 and Figure 17 show the Round Trip Time of both queues. We can see that the pFifo queue has larger RTT than the RED queue.

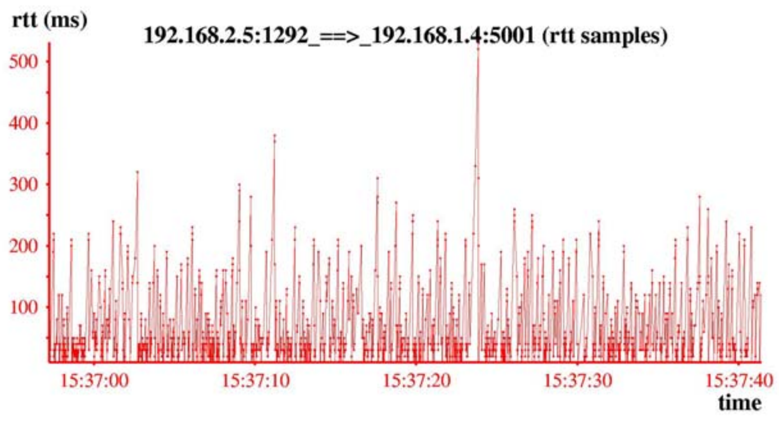


Figure 16: RTT of Flow 1

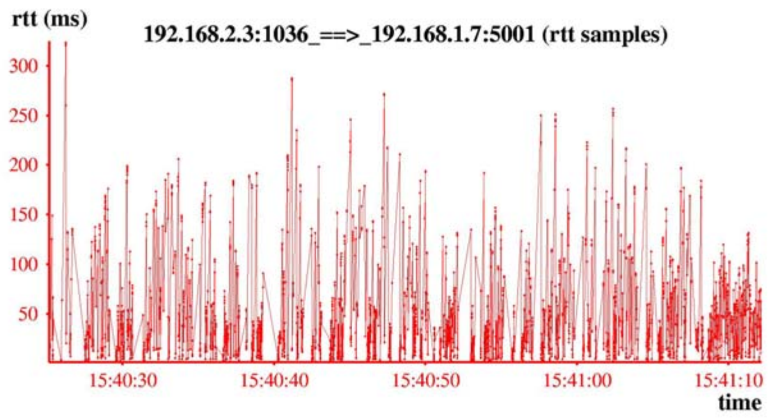


Figure 17: RTT of Flow 2

The TCP behavior on pFifo is straightforward. The TCP source sends packets based on the TCP window size and if the rate is higher than what the pFifo can sustain then the queue drops the packets. In this case we don’t have packet drops in pFifo but we do have in RED. The drops in RED queue are expected, since after a certain threshold, RED has a certain probability that start dropping packets.

Note that the RTT time on both queues varies. On pfifo the RTT is larger than the RED. The RTT time is defined by how large the queue size is and since the pFifo is a fixed size then the RTT is fixed. On the other hand, the queue size of RED queue varies based on the mean queue size.

Test 4

In test 4 we used exactly the same parameters that we used in Test 3 with the exception that the TCP window size here is 128K bytes on both ends. We have increased the TCP window size in order to increase the throughput of the TCP traffic.

Router Setup Parameters						
Source IP	Classifier	Scheduler	Class Priority	Class Weight	Filter Priority	Bandwidth
192.168.2.5	U32	CBQ	5	500K	1	5Mbps
192.168.2.3	U32	CBQ	5	500K	2	5Mbps

Table 44

Source Results									
Source IP	Destination IP	Transfer Mbytes	Rate Mbps	Frame Length	Queue Type	Queue size	Traffic Type	Packets lost	Time (s) Duration
192.168.2.5	192.168.1.4	20	3.6	1514	pfifo	50	TCP	11	44.6
192.168.2.3	192.168.1.7	20	3.4	1514	RED		TCP	21	46.7

Table 45

Table 45 shows the results. With both sources transmitting TCP traffic we get 3.6Mbps and 3.4 Mbps for flow 1 and flow 2. Here we can see that packets were dropped at the pFIFO queue. This happens since we have increased the window size of the TCP, and therefore the underlying queuing disciplines cannot handle very well situations with severe congestion.

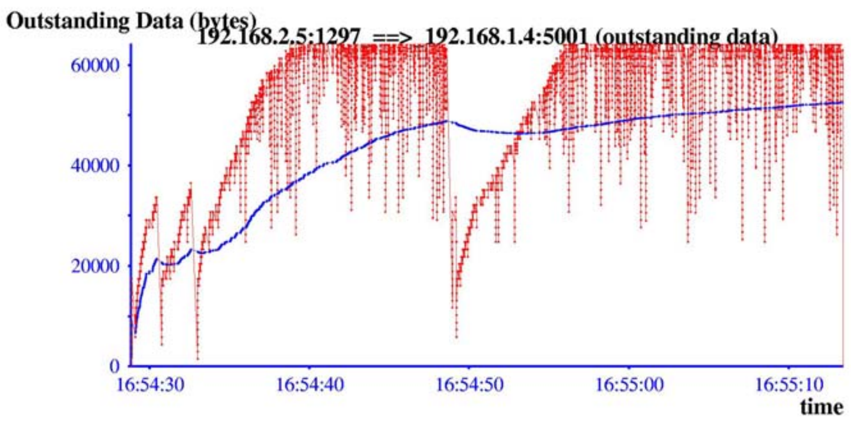


Figure 18: Outstanding Data of Flow 1

Figure 18 shows the slow start of the TCP and then shows the packets that are dropped. That's where the source starts sending at lower rates and start congestion avoidance algorithm.

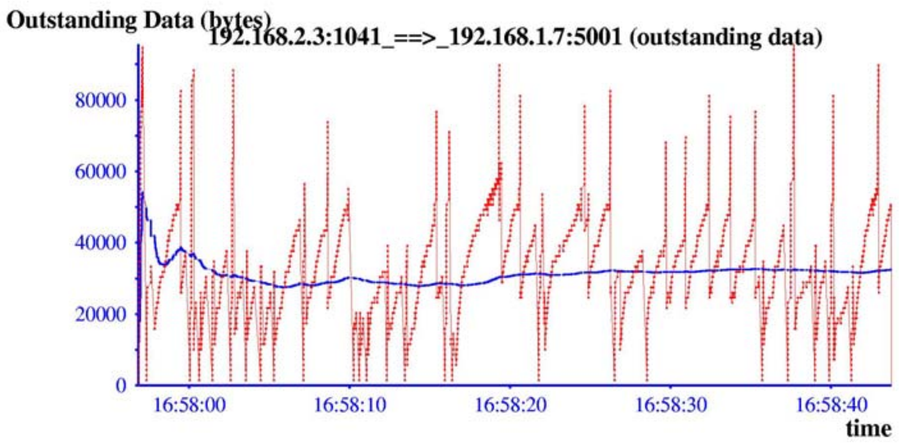


Figure 19: Outstanding Data of Flow 2

Figure 19, on the other hand, shows a different behavior. There are dropped random drops based on the probability drop of the RED queue. RED is shown to exhibit large overshoots and fluctuations over the queue.

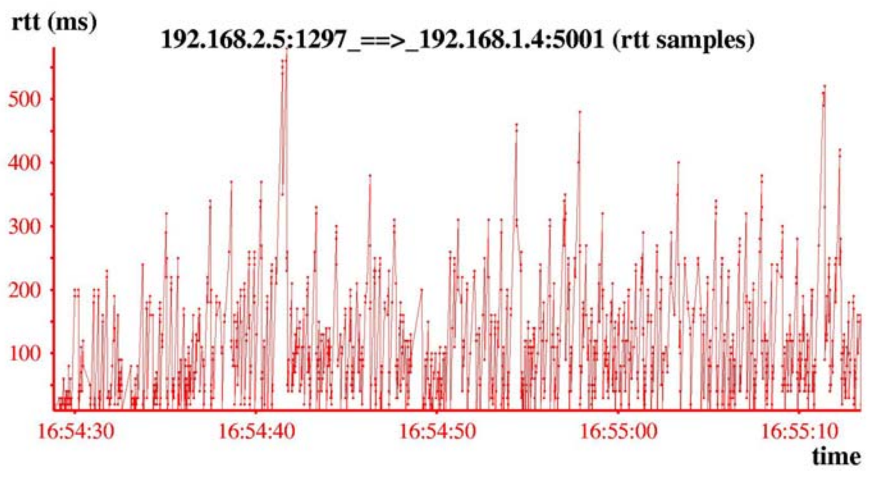


Figure 20: RTT of Flow 1

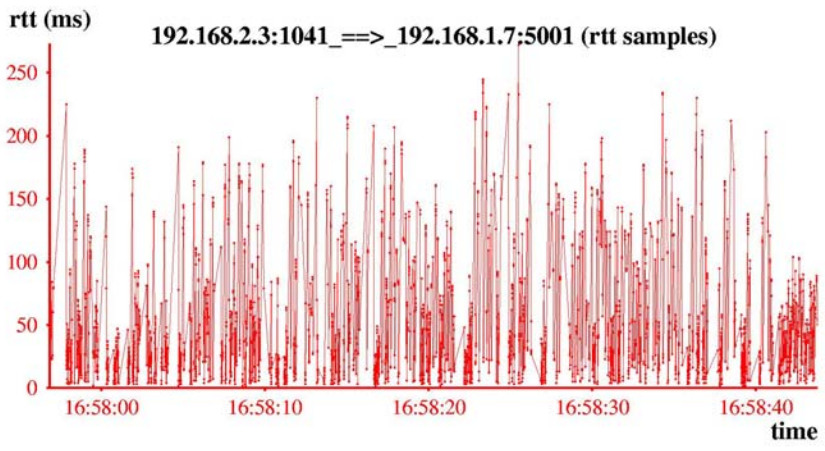


Figure 21: RTT of Flow 2

Figure 20 and Figure 21 show the RTT of the pFifo and RED queues respectively. It's obvious the RED queue get smaller RTT because the mean queue size of the queue is smaller. The maximum RTT of the RED is 250ms and the pFifo is 500ms. In both cases, there seems to be a large variation of the RTT, which indicates a lack of regulating the queues adequately.

2.2 Wireless Pilot Network

The usability and popularity of wireless communication networks has been increasing dramatically over the years, an observation, which is emphatically supported by the rapid upgrade in wireless devices and hardware capabilities. Moreover, Wireless Local Access Networks (WLANs) become more and more popular. The mobility characteristic of Wireless Networks is an innovative perspective with respect to newly developed, mobility-enhanced applications. It is required that the network level must be tested and evaluated with respect to current, existing protocols and techniques in order to explore variations and performance for different setups and settings.
Out aim is twofold:

- To set up and configure an actual (real) **wireless connectivity** in **Linux environment** and more specifically to attach a wireless extension body on an existing test network. This will be possible by extending a host at the network edge to also act as a wireless gateway. It is required that the connection of the external end host to the gateway should be done using an **ad-hoc setup**.

- To perform a number of tests and measure the performance of the wireless link. This should include existing protocols (TCP and UDP) over the wireless link. We should also take into account, the specialized conditions of the wireless link that include mobility, distance, link signal etc. Our test client should be a portable computer equipped with wireless card adaptor.

2.2.1 Tools Used for Measurements

We have acquired a number of open source useful tools to accompany our testing endeavor. These include:

- **iPerf:** This is a command line tool that allows the creation of traffic among two end-hosts. It allows creation of both TCP and UDP traffic providing a number of options including duration or size sent.

- **Ethereal Network Analyzer**: Ethereal is a network analyzing tool included with the Linux RedHat free distribution. It provides a GUI and actually allows the capturing of network traffic passing from any local machine network interface – also separating protocol packets (IP to a number of Application layer protocols.)

Finally Ethereal can save the captured result as a *tcp-dump output file*. This is a very popular file format that can also be produced with the command line *tcp-dump* tool, which works in a similar way to Ethereal.

- **Tcptrace:** This is a command line tool that receives a tcpdump output file as input and allows the extraction of useful tcp information from this file.

- **Xplot/gnuplot:** These are tools for producing graphical representations of the results. Xplot has been initially, specifically designed for network analysis plotting.

- **ACU utility from Cisco:** The utility provided with the Wireless 802.11 cards allows for some statistics reports to be collected concerning the traffic at the card as well as signal status report.

- **Other tools:** A number of tools were also tested for usability and it is believed that they are worth mentioning. These include *ettcp,* a tool similar to iperf that can also be used for creating traffic. Also, we mention *kismet* another tool for capturing network interface traffic but specifically designed for wireless networking. The advantage of kismet versus Ethereal is that it captures traffic from different wireless networks in the area. Since we only used a single network, then this was redundant but useful for future testing.

2.2.2 Network Setup

The Network setup is seen in Figure 22. We have connected one end host (Sender) on the Diff-Serv Testbed Network Hub. This computer will play the role of the traffic generator that will be sent across the wireless mean. Another computer (Gateway) was also connected to the wired network on an Ethernet interface (eth0) and was used as the wireless gateway for the Receiver portable computer using a second Ethernet interface, the Aironet PCI (eth1). The portable computer (Receiver) uses a wireless PCMCIA card as the network interface. It should connect to the wired network via the Gateway on Ad-hoc basis.

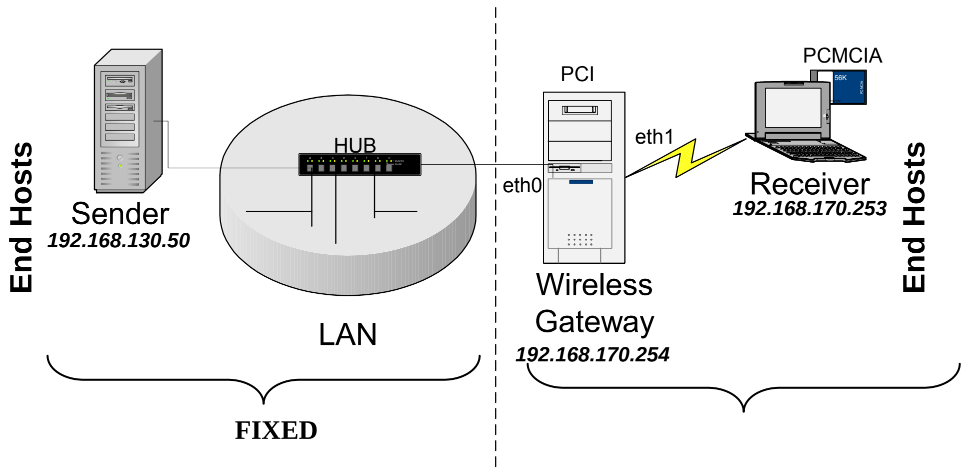


Figure 22: Network Set-up for Wireless Testbed

2.2.3 Dimensions and measures

Dimensions refer to the parameters that are subject to change during the tests. Of course multiple combinations can be done but to ensure a logical and sensible result we need to change at most two (usually just one) and contradict it to another, which we decide, that is relevant.

Measures refer to the numerical values actually recorded. These are the results of each of the combination for the dimensions. Although we may record almost all of the following measures not all will make sense for all tests while not all are applicable to all tests.

2.2.3.1 Dimensions:

- **Protocol**: We realized tests using two transport Level protocols TCP and UDP. ICMP was also used to measure some RTTs but only for checking purposes and will not appear in our following results.
- **Mobility (Signal Strength and distance)**: We measure the values as distance from the gateway antenna grows and consequently the signal strength weakens. Other issues appear here such as obstacles. These details are provided along with each test description.
- **Interval traffic is sent**: This represent the total time (usually in secs) that either TCP or UDP packets/datagrams are sent from the sender to the receiver.
- **Size of file sent**: Similarly, we can also change the size of the file sent instead of sending for a specific period.

- **Conditions / placement of devices**: We need to establish a "normal functioning environment" for our tests. This means we need to decide of a natural way to place our gateway and portable so that we will observe conditions that are more usual to such types of connections (e.g. we cannot have "line of sight" at all times!)
- **Gateway Queuing Discipline:** The gateway send packets to the wireless receiver via the wireless interface using some kind of queuing policy for outgoing packets. The "tc" command (Linux) allows us to manipulate outgoing traffic sending policy.
- **IP TOS:** The Type of Service (TOS) field in IP packets is not usually used by routers. However the default (hardware) queuing policy used with the Ethernet adapters we used is pFIFO, which does consider the TOS field. We observe the effect of the various values received by this field.

2.2.3.2 Measures:

We measure the following quantities (when applicable)

- Bandwidth (bytes per second or bits per second)
- RTT (Round Trip time) the time a packet takes to reach the receiver and back.
- Outstanding data. Since we cannot measure the TCP congestion window size at all times, measuring the outstanding data will give us a hint to the congestion conditions in the network. Here is the description provided by the *tcptrace tool* help.

  "The idea here is to estimate the congestion window as the number of unacknowledged bytes. Since we cannot accurately determine the congestion window, we use the outstanding data as an approximate of the network congestion. The outstanding data is calculated as the number of un-acknowledged bytes at any given time. For every packet received, the outstanding data at that point is calculated as the number of bytes that are as yet un-ACKed. These samples are then *weighted by the time for which they exist."*
- Time sequence (the sequence number of the packet received over time). This is a hint to let us understand the order in which packets are being retransmitted.
- For UDP traffic we may also measure
  - Packet loss
  - Jitter
- We may also record some important statistical information using the ACU tool concerning data reaching the Wireless card:

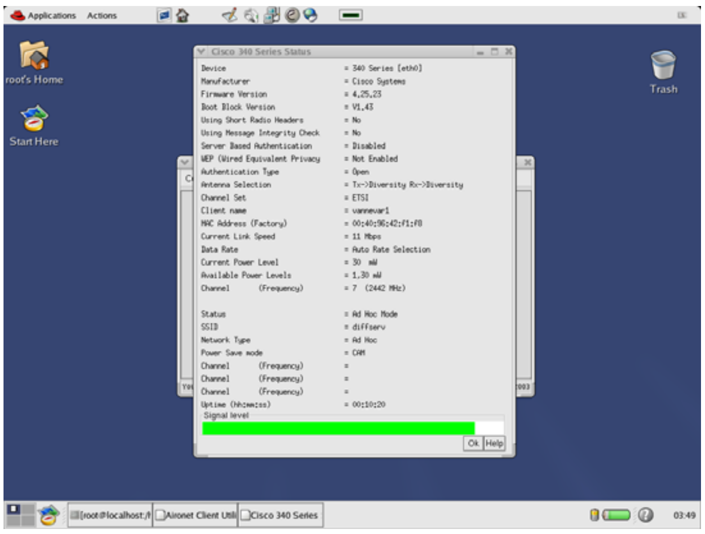
- Packets received
- Bytes received
- Duplicate packets received
- Acks transmitted

2.2.4 Evaluation of Wireless Pilot Network

Various experiments have been conducted. The wireless testbed is tested and evaluated with respect to current, existing protocols and techniques in order to explore variations and performance for different setups and settings.

2.2.4.1 Test 1 - Description: Optimal (TCP and UDP)

For our first test we place the portable computer in a "line of sight" position with respect to the gateway antenna to a distance of 4 meters, no obstacles between the two devices. Therefore we call this the optimal case (with respect to *just SIGNAL*). We notice the ACU signal report showing very high strength (see Figure 23). We also note that this is an "as-good-as-it-gets" scenario since we were unable to achieve a better signal strength even when putting the notebook next to the gateway antenna.



```
Device                              = 340 Series [eth0]
Manufacturer                        = Cisco Systems
Firmware Version                    = 4.25.23
Boot Block Version                  = V1.43
Using Short Radio Headers           = No
Using Message Integrity Check       = No
Server Based Authentication         = Disabled
WEP (Wired Equivalent Privacy       = Not Enabled
Authentication Type                 = Open
Antenna Selection                   = Tx->Diversity Rx->Diversity
Channel Set                         = ETSI
Client name                         = vannevar1
MAC Address (Factory)               = 00:40:96:42:f1:f0
Current Link Speed                  = 11 Mbps
Data Rate                           = Auto Rate Selection
Current Power Level                 = 30  mW
Available Power Levels              = 1,30 mW
Channel        (Frequency)          = 7  (2442 MHz)

Status                              = Ad Hoc Mode
SSID                                = diffserv
Network Type                        = Ad Hoc
Power Save mode                     = CAM
Channel        (Frequency)          =
Channel        (Frequency)          =
Channel        (Frequency)          =
Uptime (hh:mm:ss)                   = 00:10:20
```

Figure 23: Optimal Positioning

Under this placement we produce traffic using TCP and the UDP protocol packets / datagrams. Our purpose it to compare TCP and UDP bandwidth (throughput) as traffic flows across the wireless link. We also record a number of other statistics that concern TCP and UDP traffic that will be used as a reference point for our following tests.

NOTE: *For all tests there is no fragmentation. The packets size is 1470 bytes that fit into a single Ethernet packet. According to the 802.11 standard, a packet size of up to 2300+ bytes can be supported but still, the gateway does not know that it is sending to a Wireless client and fragments the packet when greater that 1500 bytes.*

Protocol	Interval Sending traffic	Stream Bandwidth (UDP)
TCP	60 sec	N/A
UDP	60 sec	11 Mbits/sec

Table 46: TCP and UDP setting

The following graphs (see Figure 24) show the kbits/sec throughput for the optimal scenario for TCP and UDP traffic of Table 46.

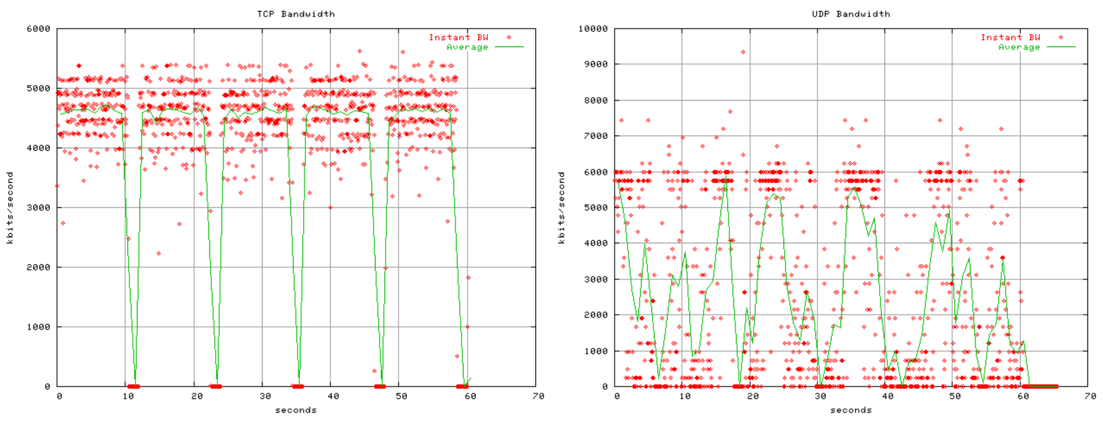


Figure 24: Optimal – TCP Vs UDP Throughput

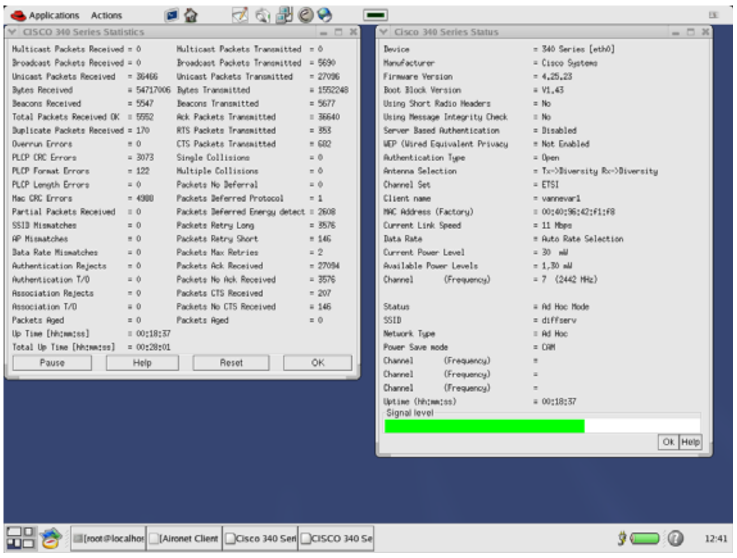
We observe a terrible behavior by UDP. Of course we need to take under consideration that we have tested the limits of UDP by creating an11Mps stream. The statistical results are shown in Table 47 below.

Protocol	Interval (s)	Total Received (MB)	Average Bandwidth (Mbps)	Packet received	Duplicate packets received	Acks Transmitted	Jitter (ms)	Loss Datagrams %
TCP	60	27	**3.76**	19552	10	19562	N/A	N/A
UDP	60	18.4	**2.56**	13159	6	13307	5.113	76

Table 47: Test 1 Statistics

2.2.4.2 Test 2 - Description: Typical (TCP and UDP)

Our next test's conditions as described will be the "standard" or "normal" conditions. In this setup we have placed the antenna and portable in non-line-of-sight position (i.e. gateway and receiver adapters do not face each other.) We keep the distance to 4 meters away in the same room. These are the usual (non-optimal) conditions for a wireless connection (see Figure 25).



```
CISCO 340 Series Statistics

Multicast Packets Received = 0        Multicast Packets Transmitted    = 0
Broadcast Packets Received = 0        Broadcast Packets Transmitted    = 5690
Unicast Packets Received   = 36466    Unicast Packets Transmitted      = 27096
Bytes Received             = 54717006 Bytes Transmitted                = 1552248
Beacons Received           = 5547     Beacons Transmitted              = 5677
Total Packets Received OK  = 5552     Ack Packets Transmitted          = 36640
Duplicate Packets Received = 170      RTS Packets Transmitted          = 353
Overrun Errors             = 0        CTS Packets Transmitted          = 682
PLCP CRC Errors            = 3073     Single Collisions                = 0
PLCP Format Errors         = 122      Multiple Collisions              = 0
PLCP Length Errors         = 0        Packets No Deferral              = 0
Mac CRC Errors             = 4988     Packets Deferred Protocol        = 1
Partial Packets Received   = 0        Packets Deferred Energy detect   = 2608
SSID Mismatches            = 0        Packets Retry Long               = 3576
AP Mismatches              = 0        Packets Retry Short              = 146
Data Rate Mismatches       = 0        Packets Max Retries              = 2
Authentication Rejects     = 0        Packets Ack Received             = 27094
Authentication T/O         = 0        Packets No Ack Received          = 3576
Association Rejects        = 0        Packets CTS Received             = 207
Association T/O            = 0        Packets No CTS Received          = 146
Packets Aged               = 0        Packets Aged                     = 0
Up Time [hh:mm:ss]         = 00:18:37
Total Up Time [hh:mm:ss]   = 00:28:01
```

```
Cisco 340 Series Status

Device                             = 340 Series [eth0]
Manufacturer                       = Cisco Systems
Firmware Version                   = 4.25.23
Boot Block Version                 = V1.43
Using Short Radio Headers          = No
Using Message Integrity Check      = No
Server Based Authentication        = Disabled
WEP (Wired Equivalent Privacy      = Not Enabled
Authentication Type                = Open
Antenna Selection                  = Tx->Diversity Rx->Diversity
Channel Set                        = ETSI
Client name                        = vannevar1
MAC Address (Factory)              = 00:40:96:42:f1:f8
Current Link Speed                 = 11 Mbps
Data Rate                          = Auto Rate Selection
Current Power Level                = 30  mW
Available Power Levels             = 1,30 mW
Channel        (Frequency)         = 7  (2442 MHz)

Status                             = Ad Hoc Mode
SSID                               = diffserv
Network Type                       = Ad Hoc
Power Save mode                    = CAM
Channel        (Frequency)         =
Channel        (Frequency)         =
Channel        (Frequency)         =
Uptime (hh:mm:ss)                  = 00:18:37
Signal level
```

Figure 25: Typical Positioning

The setting (time and steam) are the same as with Test 1. After comparing the newly reported bandwidth, we will also compare three other parameters **Round Trip Time, (RTT) Outstanding Data and Time Sequence.**

Below we present the Bandwidth graph for TCP versus UDP (see Figure 26). We now see a rather less stable (but still periodic) behaviour of TCP. We also note that there seems to be a better UDP performance that previously. This is possibly because traffic is being restricted and UPD catches up in successfully sending more datagrams through. The stats are shown below (see Table 48). Note the average bandwidth comparison and UDP loss which is now 20 units less!

Note that the scales are for TCP 0-6000kbps and UDP 0-12000kbps!

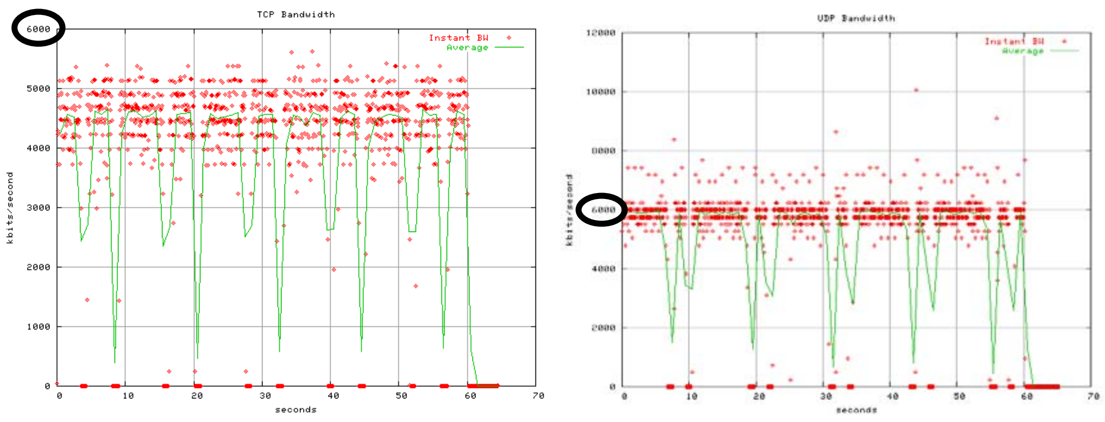
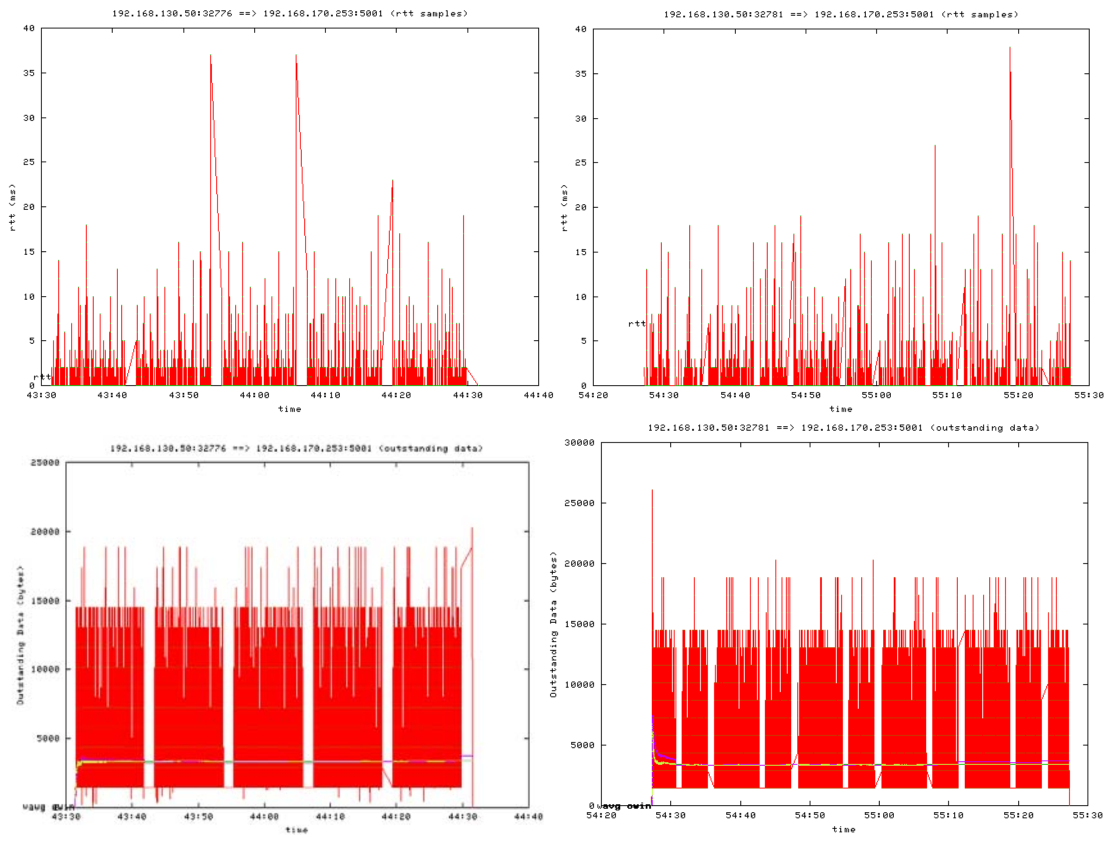


Figure 26: Typical – TCP Vs UDP Throughput

Protocol	Interval (s)	Total Received (MB)	Average Bandwidth (Mbps)	Packet received	Duplicate packets received	Acks Transmitted	Jitter (ms)	Loss Datagrams %
TCP	60	25.9	**3.62**	18789	12	18801	N/A	N/A
UDP	60	35.2	**4.90**	25133	16	27093	4.149	**54**

Table 48: Test 2 Statistics

We also present the remaining parameters comparison graphs (see Figure 27). The main observation is that bursts are rather more frequent for the typical setup. No serious changes for RTT samples graph.



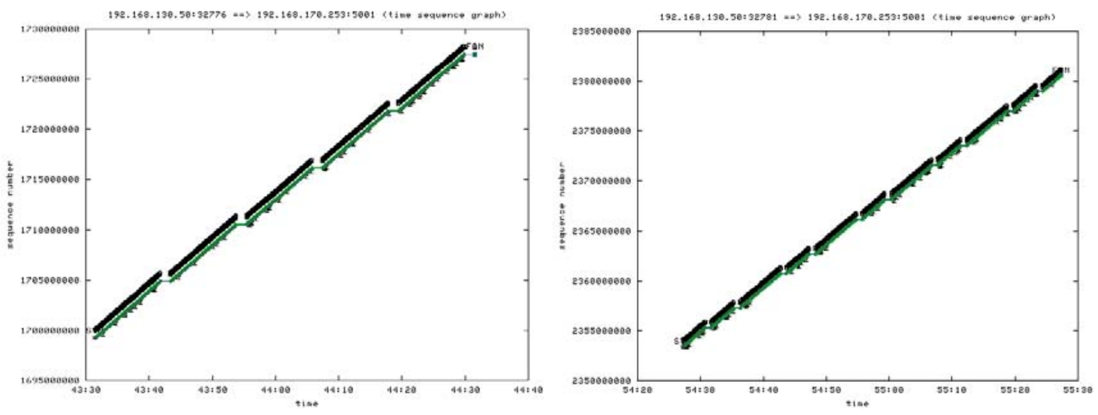
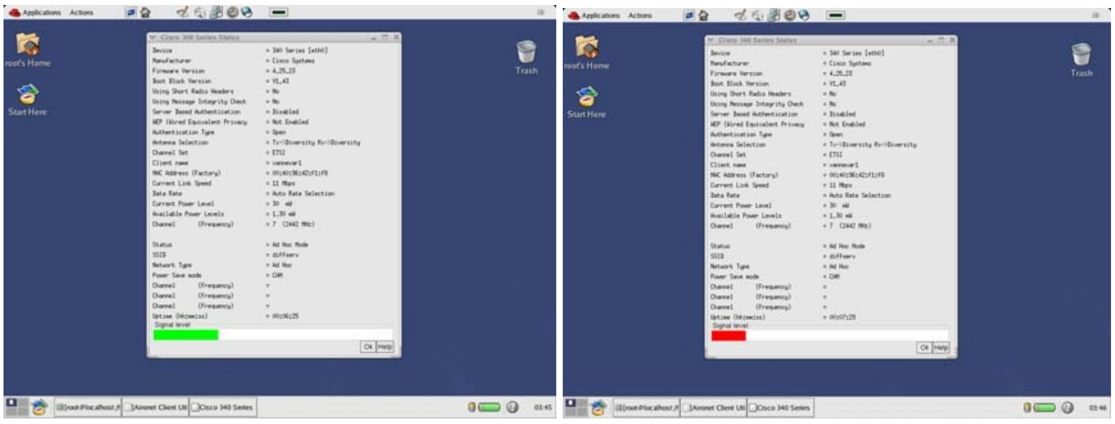


Figure27: TCP Optimal (left) Vs Typical (Right)

2.2.4.3 *Test 3 - Description: Mobility (TCP and UDP)*

In our next test we also take the mobility factor as the changing parameter. Ideally, we would have preferred to measure the effect of distance and separately, the effect of signal strength. However, it appeared to be very difficult because distance affected signal strength at almost all times.

We thus decided to keep the gateway antenna and the PCMCIA card on the receiver at a “line of sight” position at all times while moving gradually away from the gateway, observing the signal that was weakening. We cover a distance of about 60 meters, straight line, in a total time of 60 seconds, that is, we keep the conditions of the test as with tests 1 and 2 – just introduce mobility (see Figure 28). We measure both TCP and UDP performance. The stats are shown below (see Table 49).



```
Device                              = 340 Series [eth0]
Manufacturer                        = Cisco Systems
Firmware Version                    = 4.25.23
Boot Block Version                  = V1.43
Using Short Radio Headers           = No
Using Message Integrity Check       = No
Server Based Authentication         = Disabled
WEP (Wired Equivalent Privacy       = Not Enabled
Authentication Type                 = Open
Antenna Selection                   = Tx->Diversity Rx->Diversity
Channel Set                         = ETSI
Client name                         = vannevar1
MAC Address (Factory)               = 00:40:96:42:F1:F8
Current Link Speed                  = 11 Mbps
Data Rate                           = Auto Rate Selection
Current Power Level                 = 30  mW
Available Power Levels              = 1,30 mW
Channel          (Frequency)        = 7  (2442 MHz)

Status                              = Ad Hoc Mode
SSID                                = diffserv
Network Type                        = Ad Hoc
Power Save mode                     = CAM
Channel          (Frequency)        =
Channel          (Frequency)        =
Channel          (Frequency)        =
Uptime (hh:mm:ss)                   = 00:06:25
Signal level
```

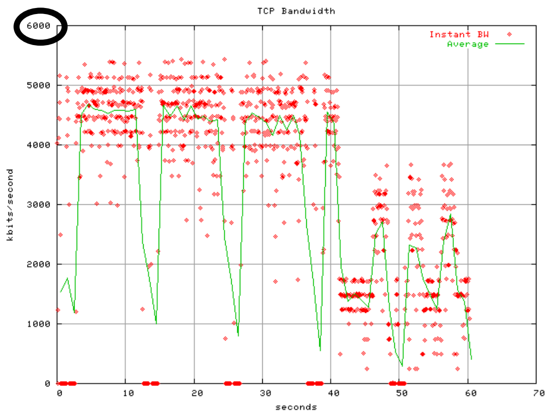
```
Device                              = 340 Series [eth0]
Manufacturer                        = Cisco Systems
Firmware Version                    = 4.25.23
Boot Block Version                  = V1.43
Using Short Radio Headers           = No
Using Message Integrity Check       = No
Server Based Authentication         = Disabled
WEP (Wired Equivalent Privacy       = Not Enabled
Authentication Type                 = Open
Antenna Selection                   = Tx->Diversity Rx->Diversity
Channel Set                         = ETSI
Client name                         = vannevar1
MAC Address (Factory)               = 00:40:96:42:F1:F8
Current Link Speed                  = 11 Mbps
Data Rate                           = Auto Rate Selection
Current Power Level                 = 30  mW
Available Power Levels              = 1,30 mW
Channel          (Frequency)        = 7  (2442 MHz)

Status                              = Ad Hoc Mode
SSID                                = diffserv
Network Type                        = Ad Hoc
Power Save mode                     = CAM
Channel          (Frequency)        =
Channel          (Frequency)        =
Channel          (Frequency)        =
Uptime (hh:mm:ss)                   = 00:07:29
Signal level
```

Figure 28: Signal Strength at around 40 and 50 meters

The following graph shows the TCP Vs UDP comparison (see Figure 29). We note a similar behavior but different throughput. UDP is still superior in that it sends data at almost 40% higher rate, in the expense, however, of high losses.

The mobility factor seems to have little (or at least smoothly negative) affect on bandwidth until we reach the distance of about 50 meters where we see an almost complete collapse. It is as if the device is trying hard to keep the connection up at a good level although the signal is weakening but crashes down at last.

Another observation is the **duplicate packets received increase for TCP & UDP**.



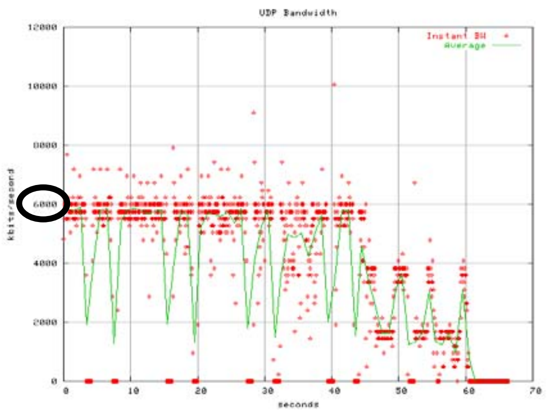
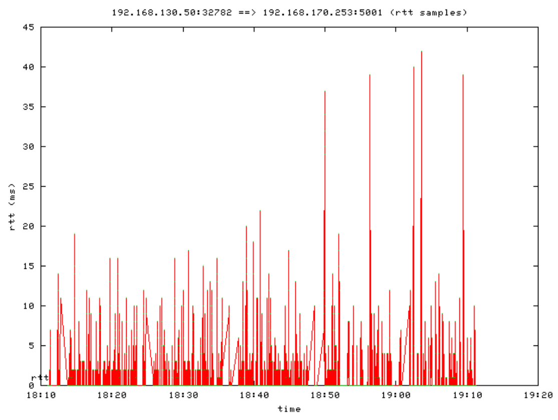


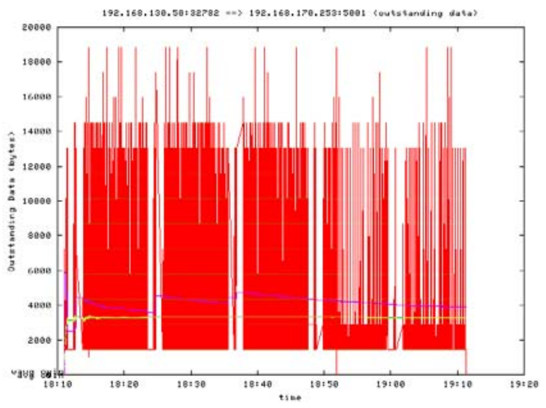
Figure 29: TCP Vs UDP at Mobility

Protocol	Interval (s)	Total Received (MB)	Average Bandwidth (Mbps)	Packet received	Duplicate packets received	Acks Transmitted	Jitter (ms)	Loss Datagrams %
TCP	60	20.4	**2.84**	14774	134	14908	N/A	N/A
UDP	60	28.3	**3.93**	20174	103	21905	11.512	63

Table 49: Test 3 Statistics

Concerning TCP, here ate the graphs for RTT, Outstanding Data and Sequence (see Figure 30).





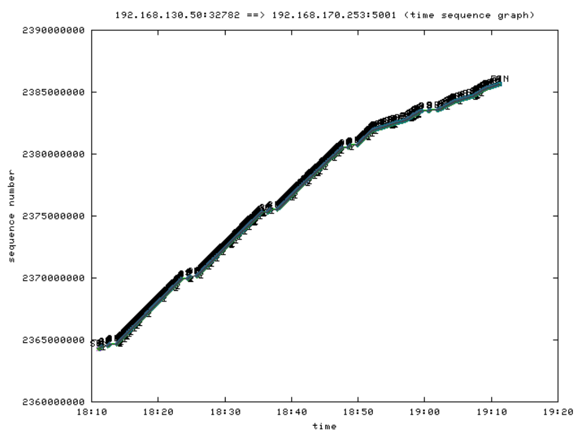


Figure 30: Test 3 RTT, Outstanding Data and Sequence graphs

2.2.4.4 Test 4 - Description: Effect of ToS

The default queuing discipline applied and (hardware set) at our Ethernet 802.11 cards was Priority-Fist-In-Fist-Out-Fast (pfifo_fast). Although this is still a FIFO queue, it does consider the TOS IP field for forwarding outgoing traffic. We have changed the actual values of TOS in packages sent, and observe the behavior of the system, under the **"Typical" conditions** of Test 2. The test had two dimensions changed:

- Sending traffic for a specific interval (30 sec) – Constant Time
- Sending a specific size file (4MB) – Constant Load

Traffic was generated from four different clients simultaneously sending traffic to the gateway on their way to the receiver. Each client had a different TOS bit set and another one had none bit set (i.e. the default.)

The TOS field is usually ignored by routers. However, it was not ignored for the pFIFO queue used at the gateway to forward packets through the wireless link.

```
     0     1     2     3     4     5     6     7
  +-----+-----+-----+-----+-----+-----+-----+-----+
  |                 |                       |     |
  |   PRECEDENCE    |          TOS          | MBZ |
  |                 |                       |     |
  +-----+-----+-----+-----+-----+-----+-----+-----+
```

Here is what each bit of the TOS field means

Binary	**Decimcal**	**Meaning**
1000	8	Minimize delay (md)
0100	4	Maximize throughput (mt)
0010	2	Maximize reliability (mr)
0001	1	Minimize monetary cost (mmc) – NOT USED
0000	0	Normal Service

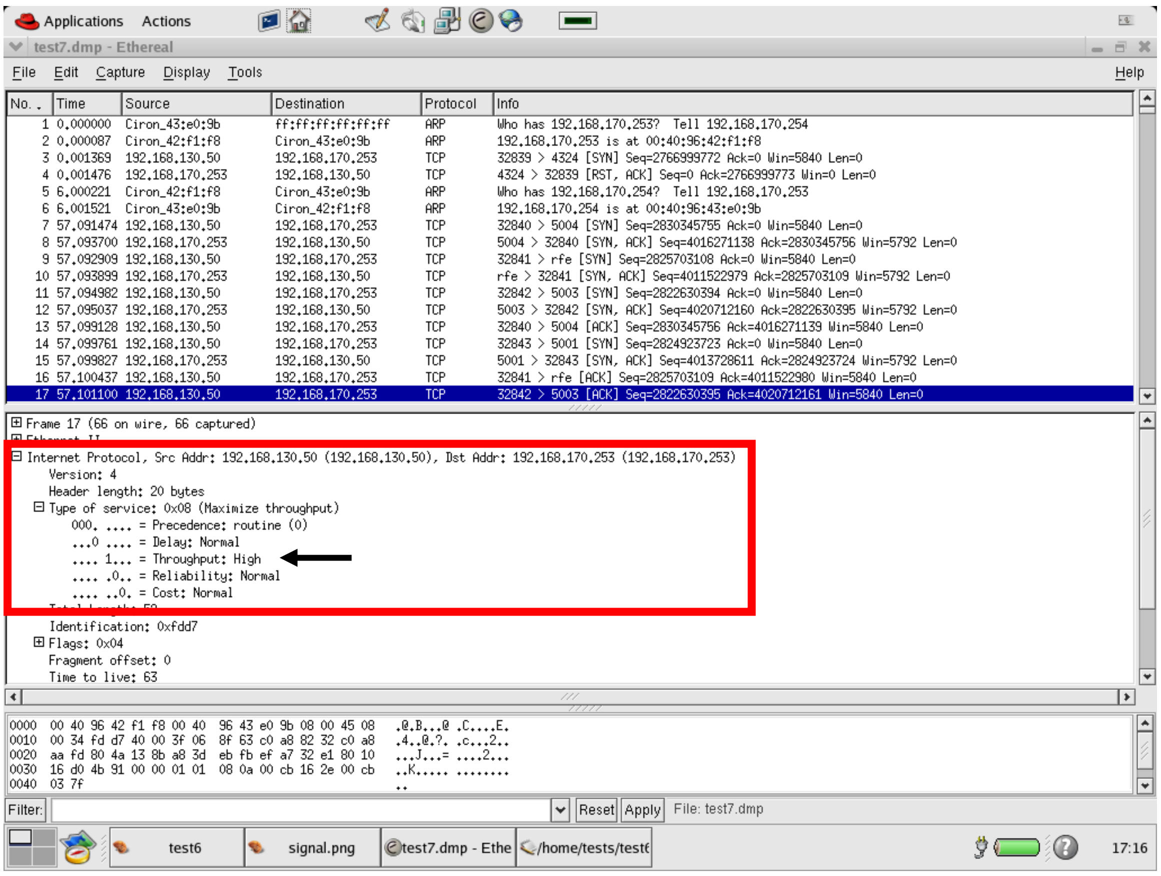
We ignored the “monetary cost” bit and was not used for our test. It does not have any special meaning for our results.

The pfifo_fast queue is, as the name says, First In, First Out, which means that no packet receives special treatment. ***At least, not quite.*** This queue has 3 so called **'bands'**. Within each band, FIFO rules apply. However, as long as there are packets waiting in band 0, band 1 won't be processed. Same goes for band 1 and band 2.The kernel honors the so called Type of Service flag of packets, and takes care to insert 'minimum delay' packets in band 0.

Next we show how the default pFIFO fast queue is handling packet.

TOS	**Bits**	**Means**	**Linux Priority**	**Band**
0x0	0	Normal Service	0 Best Effort	1
0x2	1	Minimize Monetary Cost	1 Filler	2
0x4	2	Maximize Reliability	0 Best Effort	1
0x6	3	mmc+mr	0 Best Effort	1
0x8	4	Maximize Throughput	2 Bulk	2
0xa	5	mmc+mt	2 Bulk	2
0xc	6	mr+mt	2 Bulk	2
0xe	7	mmc+mr+mt	2 Bulk	2
0x10	8	Minimize Delay	6 Interactive	0
0x12	9	mmc+md	6 Interactive	0
0x14	10	mr+md	6 Interactive	0
0x16	11	mmc+mr+md	6 Interactive	0
0x18	12	mt+md	4 Int. Bulk	1
0x1a	13	mmc+mt+md	4 Int. Bulk	1
0x1c	14	mr+mt+md	4 Int. Bulk	1
0x1e	15	mmc+mr+mt+md	4 Int. Bulk	1

A script was created which started the four senders almost simultaneously. Therefore we consider four separate but simultaneous TCP connections. We see in Figure 31, as packets are received by Ethereal how we can check out the TOS bits (set and not set.)



```
No. . Time      Source            Destination        Protocol  Info
    1 0.000000  Ciron_43:e0:9b    ff:ff:ff:ff:ff:ff  ARP       Who has 192.168.170.253?  Tell 192.168.170.254
    2 0.000087  Ciron_42:f1:f8    Ciron_43:e0:9b     ARP       192.168.170.253 is at 00:40:96:42:f1:f8
    3 0.001369  192.168.130.50    192.168.170.253    TCP       32839 > 4324 [SYN] Seq=2766999772 Ack=0 Win=5840 Len=0
    4 0.001476  192.168.170.253   192.168.130.50     TCP       4324 > 32839 [RST, ACK] Seq=0 Ack=2766999773 Win=0 Len=0
    5 6.000221  Ciron_42:f1:f8    Ciron_43:e0:9b     ARP       Who has 192.168.170.254?  Tell 192.168.170.253
    6 6.001521  Ciron_43:e0:9b    Ciron_42:f1:f8     ARP       192.168.170.254 is at 00:40:96:43:e0:9b
    7 57.091474 192.168.130.50    192.168.170.253    TCP       32840 > 5004 [SYN] Seq=2830345755 Ack=0 Win=5840 Len=0
    8 57.093700 192.168.170.253   192.168.130.50     TCP       5004 > 32840 [SYN, ACK] Seq=4016271138 Ack=2830345756 Win=5792 Len=0
    9 57.092909 192.168.130.50    192.168.170.253    TCP       32841 > rfe [SYN] Seq=2825703108 Ack=0 Win=5840 Len=0
   10 57.093899 192.168.170.253   192.168.130.50     TCP       rfe > 32841 [SYN, ACK] Seq=4011522979 Ack=2825703109 Win=5792 Len=0
   11 57.094982 192.168.130.50    192.168.170.253    TCP       32842 > 5003 [SYN] Seq=2822630394 Ack=0 Win=5840 Len=0
   12 57.095037 192.168.170.253   192.168.130.50     TCP       5003 > 32842 [SYN, ACK] Seq=4020712160 Ack=2822630395 Win=5792 Len=0
   13 57.099128 192.168.130.50    192.168.170.253    TCP       32840 > 5004 [ACK] Seq=2830345756 Ack=4016271139 Win=5840 Len=0
   14 57.099761 192.168.130.50    192.168.170.253    TCP       32843 > 5001 [SYN] Seq=2824923723 Ack=0 Win=5840 Len=0
   15 57.099827 192.168.170.253   192.168.130.50     TCP       5001 > 32843 [SYN, ACK] Seq=4013728611 Ack=2824923724 Win=5792 Len=0
   16 57.100437 192.168.130.50    192.168.170.253    TCP       32841 > rfe [ACK] Seq=2825703109 Ack=4011522980 Win=5840 Len=0
   17 57.101100 192.168.130.50    192.168.170.253    TCP       32842 > 5003 [ACK] Seq=2822630395 Ack=4020712161 Win=5840 Len=0
```

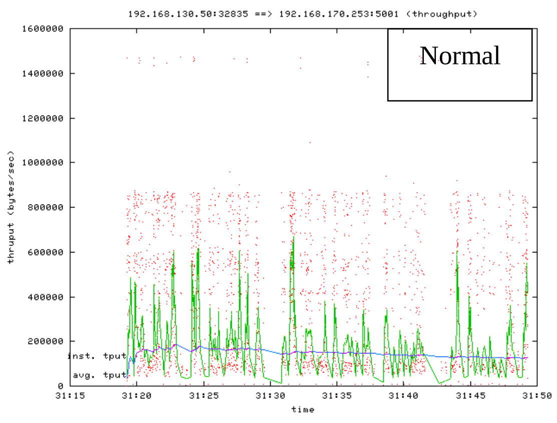
```
⊞ Frame 17 (66 on wire, 66 captured)
⊞ Ethernet II
⊟ Internet Protocol, Src Addr: 192.168.130.50 (192.168.130.50), Dst Addr: 192.168.170.253 (192.168.170.253)
    Version: 4
    Header length: 20 bytes
  ⊟ Type of service: 0x08 (Maximize throughput)
      000. .... = Precedence: routine (0)
      ...0 .... = Delay: Normal
      .... 1... = Throughput: High
      .... .0.. = Reliability: Normal
      .... ..0. = Cost: Normal
    Total Length: 52
    Identification: 0xfdd7
  ⊞ Flags: 0x04
    Fragment offset: 0
    Time to live: 63
```

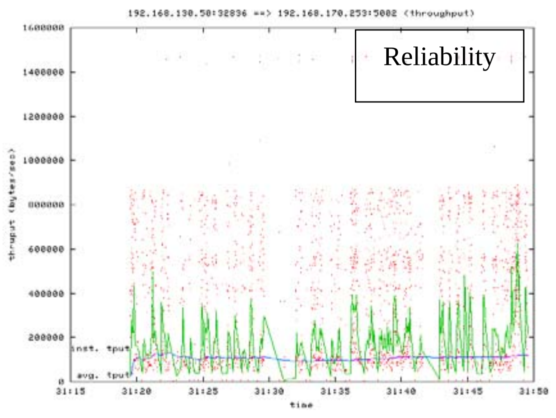
```
0000  00 40 96 42 f1 f8 00 40  96 43 e0 9b 08 00 45 08   .@.B...@ .C....E.
0010  00 34 fd d7 40 00 3f 06  8f 63 c0 a8 82 32 c0 a8   .4..@.?. .c...2..
0020  aa fd 80 4a 13 8b a8 3d  eb fb ef a7 32 e1 80 10   ...J...= ....2...
0030  16 d0 4b 91 00 00 01 01  08 0a 00 cb 16 2e 00 cb   ..K..... ........
0040  03 7f                                              ..
```

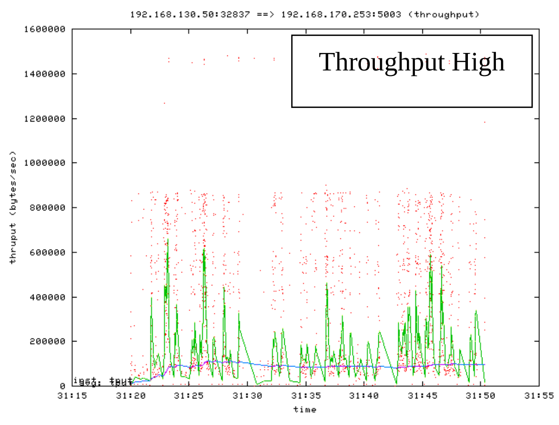
Figure 31: An example of TOS bit set shown in Ethereal

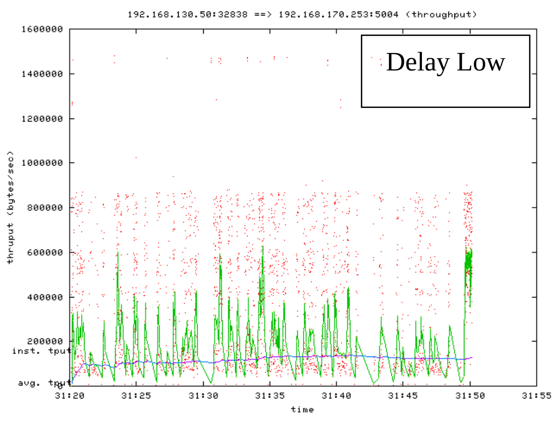
The following graphs (see Figure 32-33) show the comparison among the four “traffic classes” and is noted which is which. Table 50-51 shows some important statistics concerning the test. Unfortunately we were unable to separate statistics shown previously (such as total number of packets and duplicates) since ACU reports the total amounts (i.e. aggregate for all four connections.)

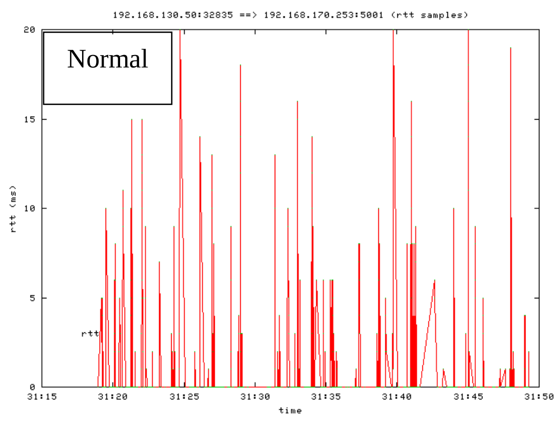
2.2.4.4.1   Test 4-A – Constant Time

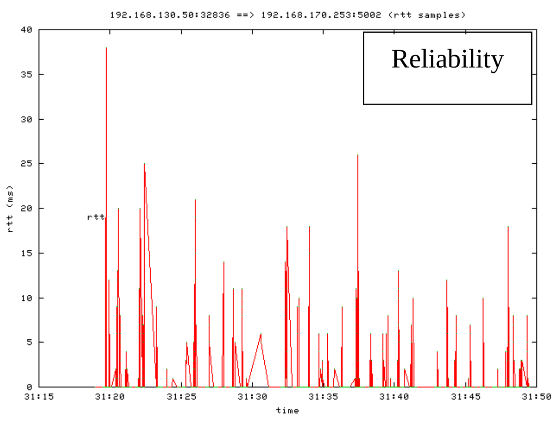


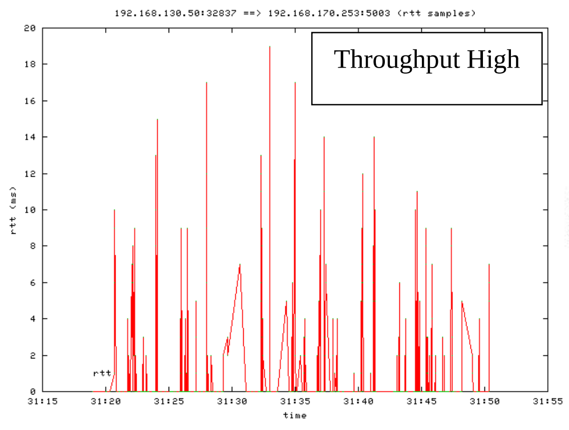


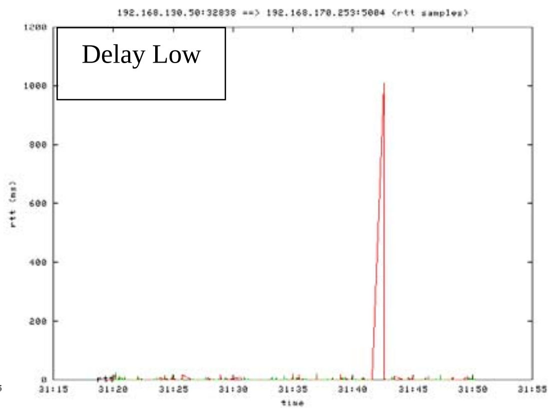

192.168.130.50:32837 ==> 192.168.170.253:5003 (throughput)
Throughput High
thruput (bytes/sec)
time
inst. tput
avg. tput

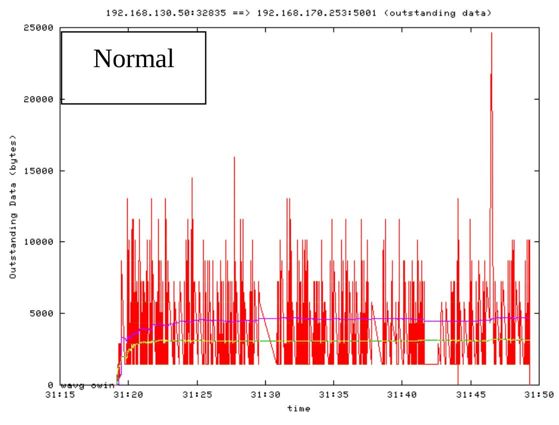

192.168.130.50:32838 ==> 192.168.170.253:5004 (throughput)
Delay Low
thruput (bytes/sec)
time
inst. tput
avg. tput

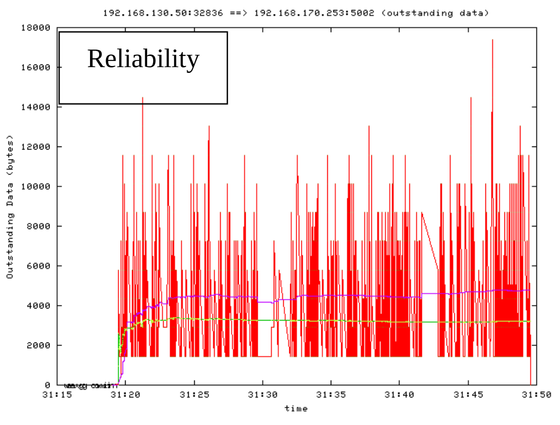

192.168.130.50:32835 ==> 192.168.170.253:5001 (rtt samples)
Normal
rtt (ms)
time
rtt


192.168.130.50:32836 ==> 192.168.170.253:5002 (rtt samples)
Reliability
rtt (ms)
time
rtt


192.168.130.50:32837 ==> 192.168.170.253:5003 (rtt samples)
Throughput High
rtt (ms)
time
rtt


192.168.130.50:32838 ==> 192.168.170.253:5004 (rtt samples)
Delay Low
rtt (ms)
time


192.168.130.50:32835 ==> 192.168.170.253:5001 (outstanding data)
Normal
Outstanding Data (bytes)
time
wavg owin


192.168.130.50:32836 ==> 192.168.170.253:5002 (outstanding data)
Reliability
Outstanding Data (bytes)
time
wavg owin

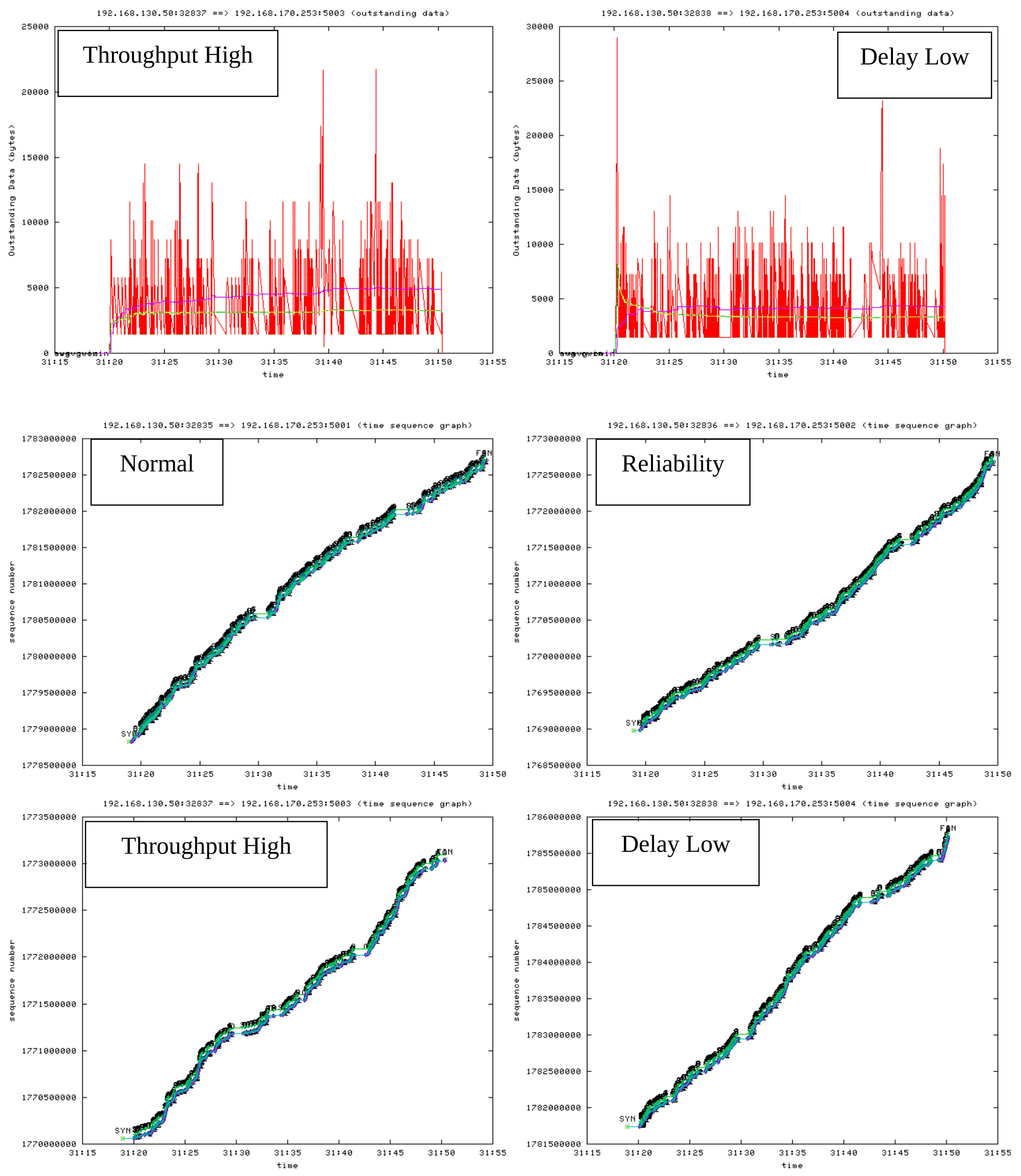
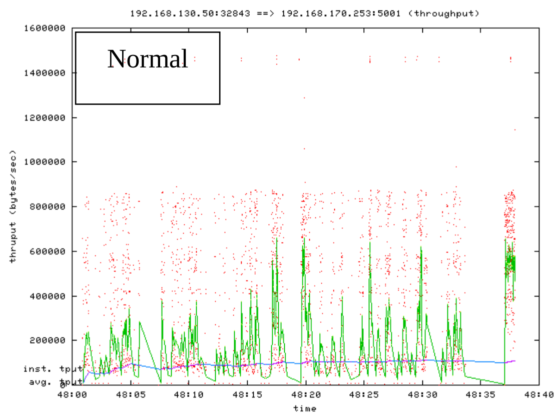


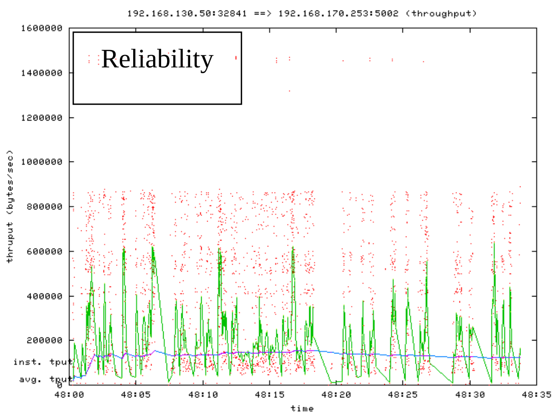
Figure 32: Test 4-A Graphs

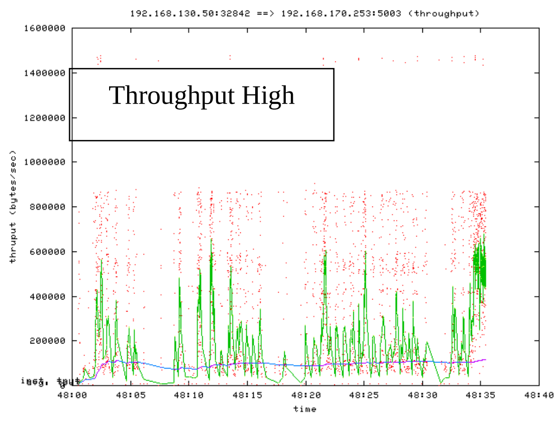
TOS	Interval (s)	Total Received (MB)	Average Bandwidth (Mbps)
0x0	30	3.69	**1.03**
0x4	30	3.54	**0.988**
0x8	30	2.85	**0.789**
0x10	30	3.81	**1.06**

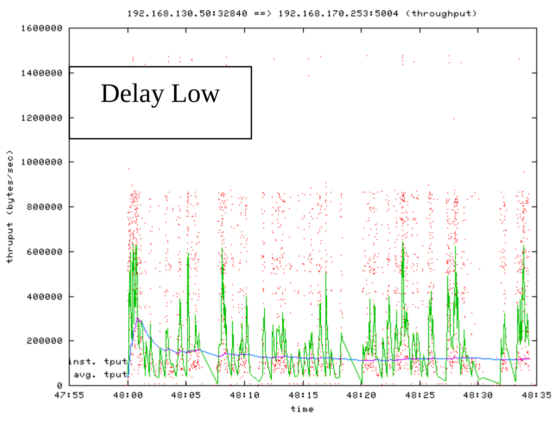
Table 50: Test 4-A Statistics

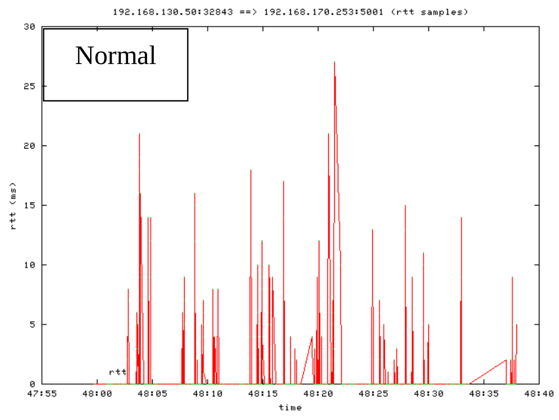
2.2.4.4.2 Test 4-B – Constant Load

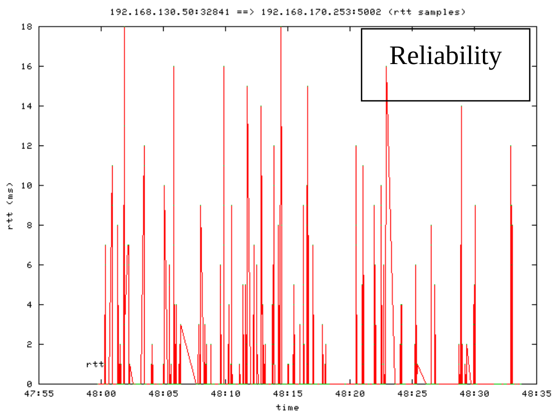


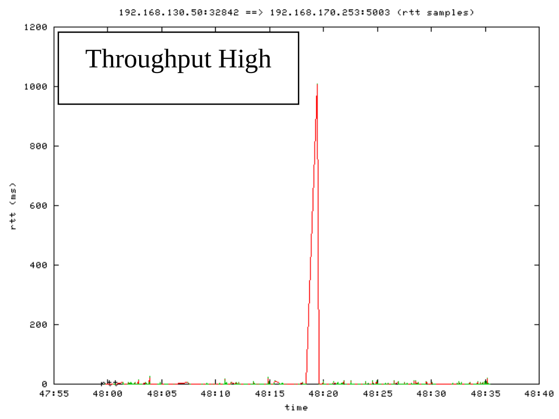


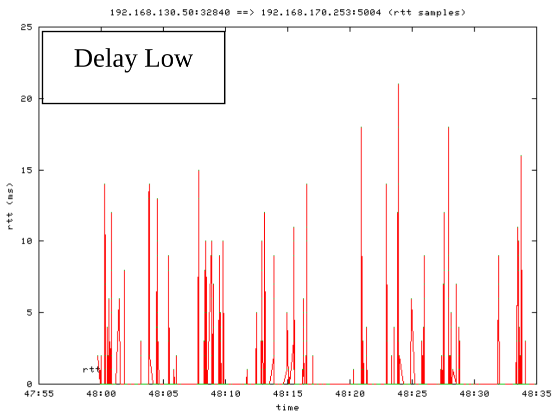












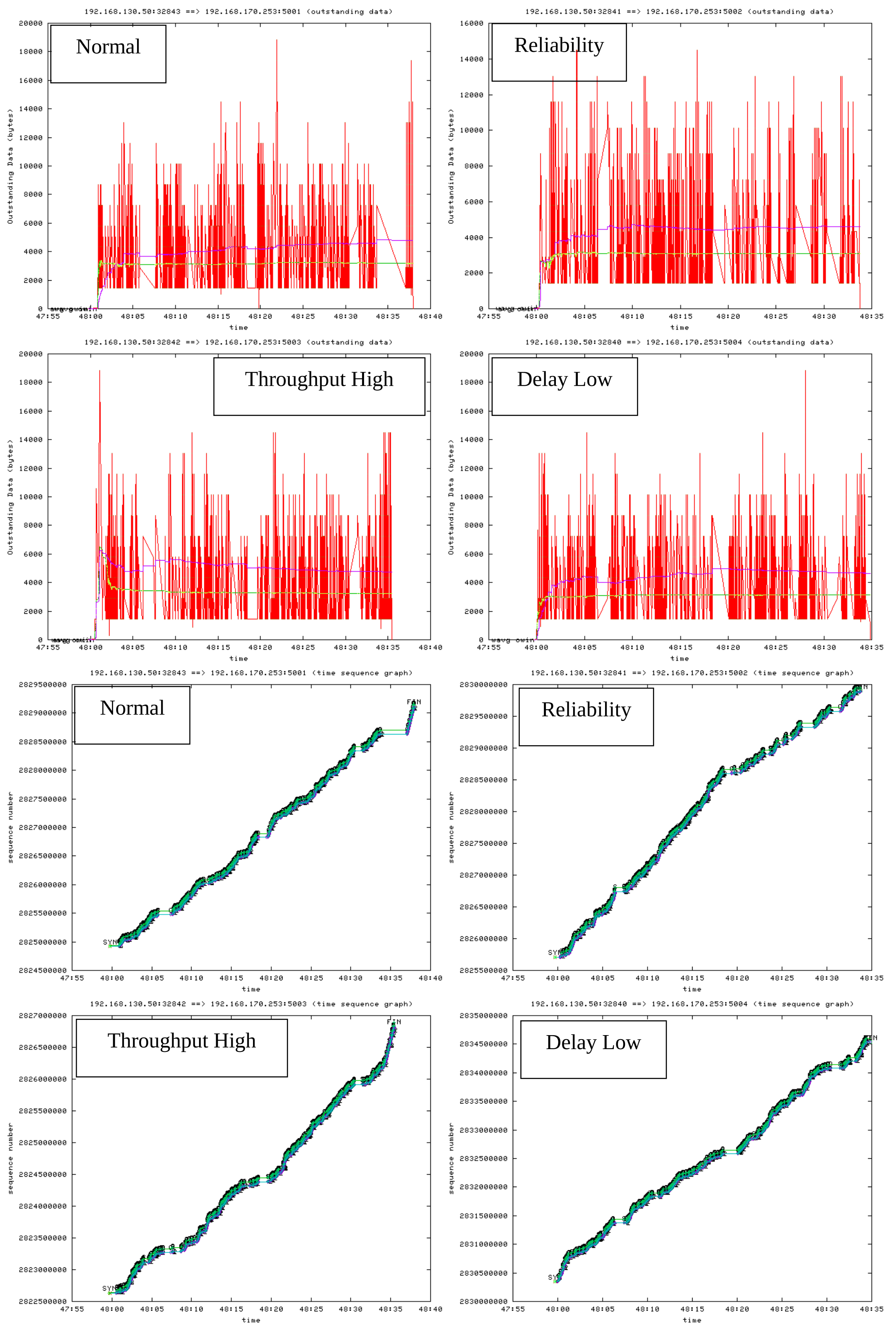


Figure 33: Test 4-B graphs

TOS	Interval (s)	Total Received (MB)	Average Bandwidth (Mbps)
0x0	37.0	4	**906**
0x4	34.9	4	**962**
0x8	34.9	4	**962**
0x10	33.5	4	**1.00**

Table 51: Test 4-B Statistics

2.2.4.5 Test 5 - Description: Gateway Queuing Disciplines

Our last test investigates whether the gateway sending queuing discipline might affect the performance at the wireless link (see Figure 34). We have tested three alternative queuing disciplines which we briefly describe below.

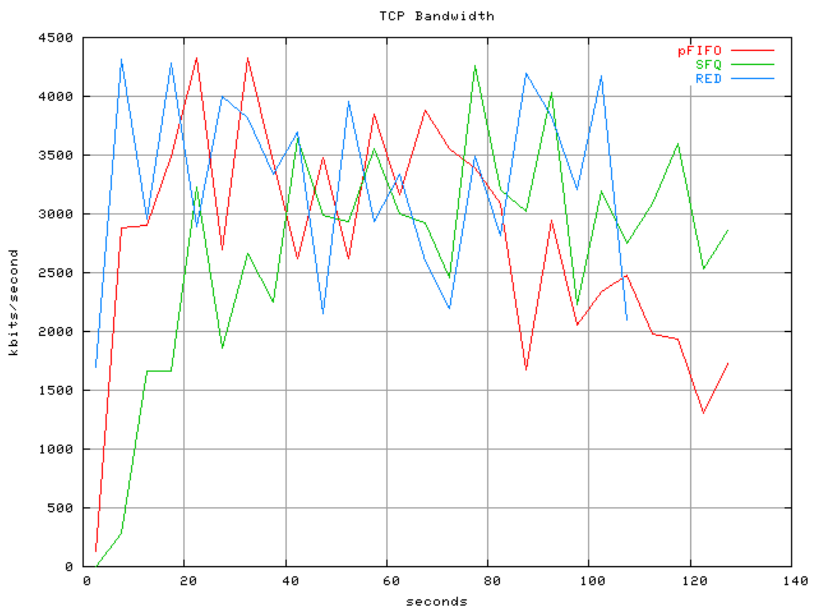
We also use the following test conditions for all three tests:

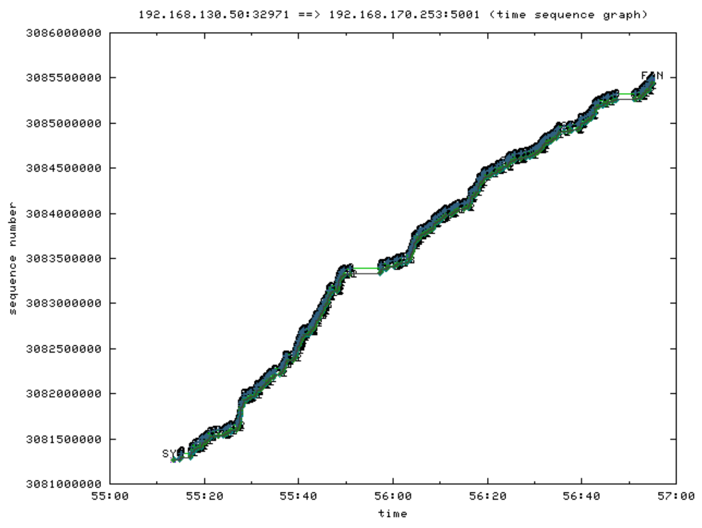
- Number of simultaneous sending threads: **10**
- File Size Sent (per thread): **4MB**

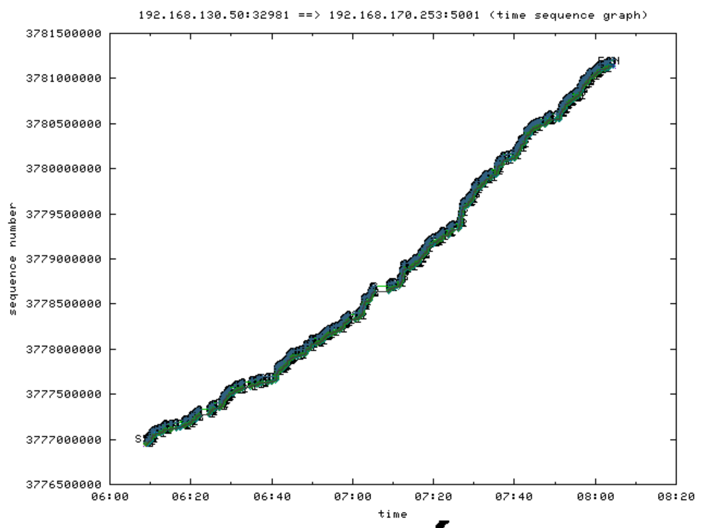
We selected to send TCP traffic. The number of threads was large enough to produce some congestion at the gateway since threads will run simultaneously. Instead of time we select to send a file of specified size to force TCP "deliver" the load and not to allow packets not to be transmitted. For RED queue there are some additional parameters, as shown below.

Min (bytes)	Max (bytes)	probability	Limit (bytes)	burst	Avpkt (bytes)	bandwidth	ecn
30000 (30 packets)	90000 (90 packets)	0.02	100000 (100 packets)	50	10000	NOT SET	NOT SET

NOTE: ***All queue disciplines were added to the interface as root (i.e. no internal classes, no CBQ usage).***


TCP Bandwidth
kbits/second
seconds
pFIFO
SFQ
RED
0
500
1000
1500
2000
2500
3000
3500
4000
4500
0
20
40
60
80
100
120
140


192.168.130.50:32971 ==> 192.168.170.253:5001 (time sequence graph)
sequence number
time
3081000000
3081500000
3082000000
3082500000
3083000000
3083500000
3084000000
3084500000
3085000000
3085500000
3086000000
55:00
55:20
55:40
56:00
56:20
56:40
57:00
SYN
FIN



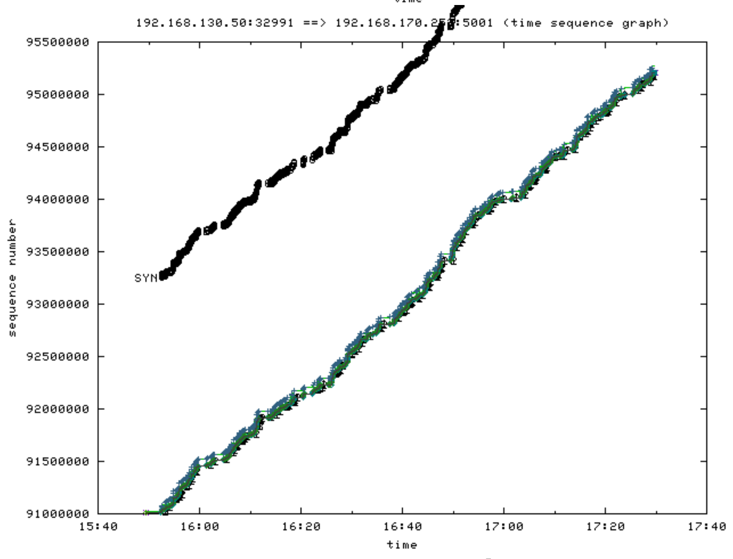


Figure 34: Test 5 graphs

3 Simulations

3.1 Use of Simulator tool for the performance evaluation of networks

We mainly concentrate on a Differentiated Services environment for providing quality of service in IP networks. The use of a simulator can help in the investigation and performance evaluation of existing congestion control algorithms and active queue management schemes. Furthermore, the analysis of simulation results can lead to improvement and development of new QoS mechanisms.

The search for the right simulation environment has followed some basic guidelines as there are many “state-of-the-art” simulation environments available, both publicly and commercially. A major commercially available network simulator is OPNET [4]. On the other hand, a major non-commercially, publicly available network simulator is NS-2 [5]. Criteria and guidelines followed for the selection of the simulation environment include the ability of the simulator to provide: granularity in models, protocol model richness, dynamic definition of network topology, user-friendly programming model, debugging and tracing support, widely accepted efficient performance, and source availability. The simulator itself has to be user-friendly, must provide a hierarchical architecture to ensure flexibility, and should have a large and active user community. Based on these criteria, NS-2 - a non-commercial, publicly available, open source, object-oriented simulator written in C++ with an OTcl interpreter as a front-end - is chosen.

NS-2 is publicly available whereas OPNET is commercially available on a yearly renewable contract basis. As a public domain simulator, NS-2 has a large user community and is widely recognized and accepted as an efficient and accurate network simulation tool. There is a high possibility that a large population of users will validate the simulation models developed as part of ENDIKTIS since it is common practice to publish simulation scripts. One of the main objectives of NS-2 is to provide a collaborative network simulation environment. It is freely distributed and open source, hence, allowing the sharing of code, protocols, and models. This accommodates the comparison and evaluation of competing protocols and models that are under consideration. Collaboration among NS-2 users results in an increased confidence in the simulation results obtained since more people investigate and analyse the simulation models developed.

NS-2 is an object-oriented simulator written in C++ with an OTcl interpreter as a front-end. The reason for using two programming languages is flexibility. Detailed simulations of protocols require a system programming language which can efficiently manipulate bytes and packet headers, and implement algorithms that run over large data sets. C++ is fast to run but slower to change. Therefore, it is suitable for detailed protocol implementation. A large part of network research involves fine-tuning certain parameters and configurations and quickly exploring different scenarios of interest. In this case, OTcl is used as it can be changed very quickly and interactively. The tradeoff for this convenience is longer simulation times.

Inside the ENDIKTIS group, NS-2 is already used as the simulation tool for many research activities. Consequently, there exists more expertise in NS-2 compared to OPNET. Based on these grounds, and after evaluating the comparative advantages of the network simulation tools currently available, NS-2 has been chosen as the most appropriate simulation environment for the ENDIKTIS project.

3.2 Preliminary simulation results

We have conducted some preliminary simulations for the evaluation of simple topologies-scenarios with the use of the network simulator NS-2. Scenarios 1-3 include simple network topologies, where the sources send packets with a constant rate using the UDP transport protocol. Scenario 4 uses a simple network topology that supports differentiated services with TCP/FTP traffic. Through these scenarios we measure the throughput of the bottleneck link.

3.2.1 Scenario 1

The network topology used for Scenario 1 is shown in Figure 35. It can be seen that a source send packets with a constant bit rate of 2Mbit/sec to a destination through a single router. The bottleneck link capacity is set to 3 Mbit/sec. The router uses Tail Drop (that is, FIFO) queuing discipline / packet drop mechanism. The buffer size is set to 200 packets, whereas the packet size is 1000 bytes.

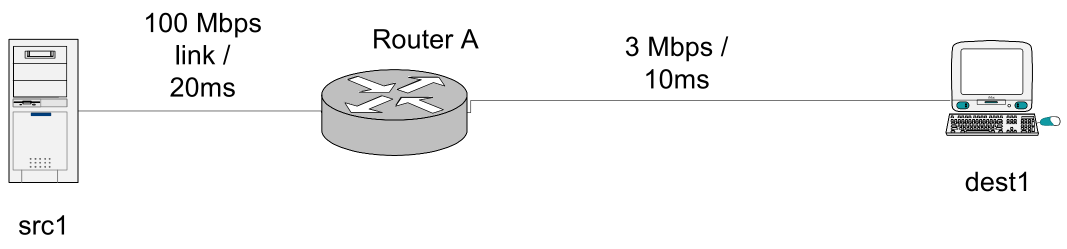


Figure 35. Scenario 1: Network topology

Figure 36 shows the throughput of the bottleneck link. It can be observed that the destination receives all the packets sent by the source successfully with no loss.

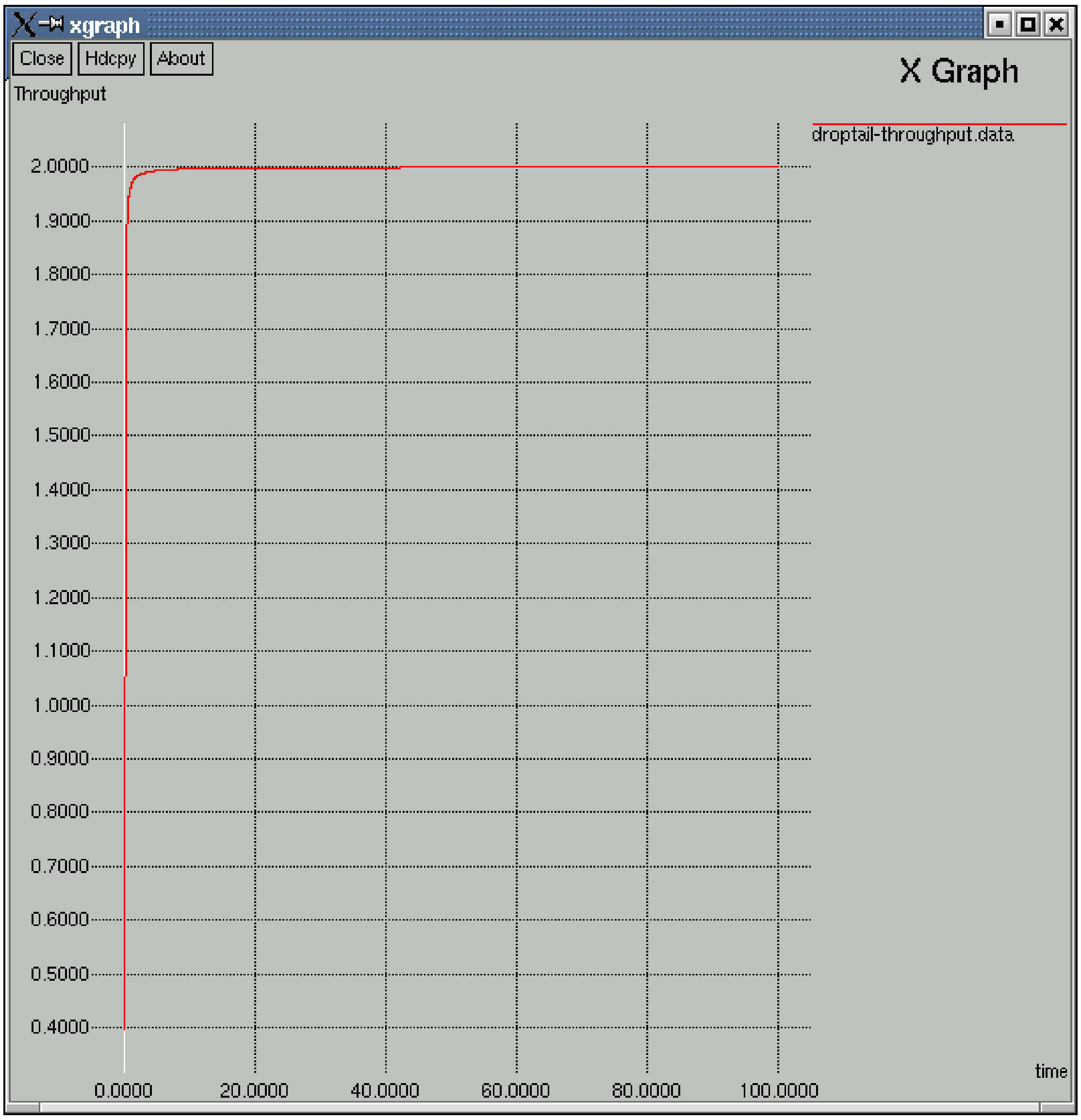


Figure 36: Scenario 1: Throughput

3.2.2 Scenario 2

The network topology used for Scenario 2 is shown in Figure 37. We have used Scenario 1 with the addition of one extra source that sends traffic with a constant bit rate of 4 Mbit/sec to the same destination. From Figure 38, we can conclude that there are packet losses from both sources. The maximum throughput for both sources equals the bottleneck link capacity, as expected.

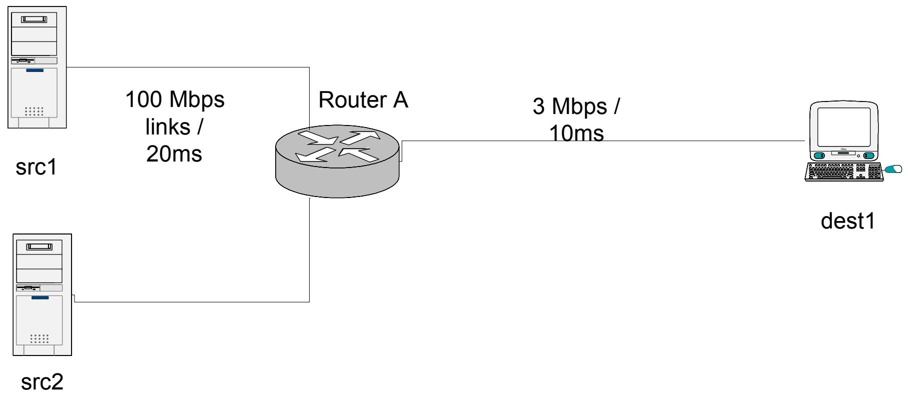


Figure 37. Scenario 1: Network topology

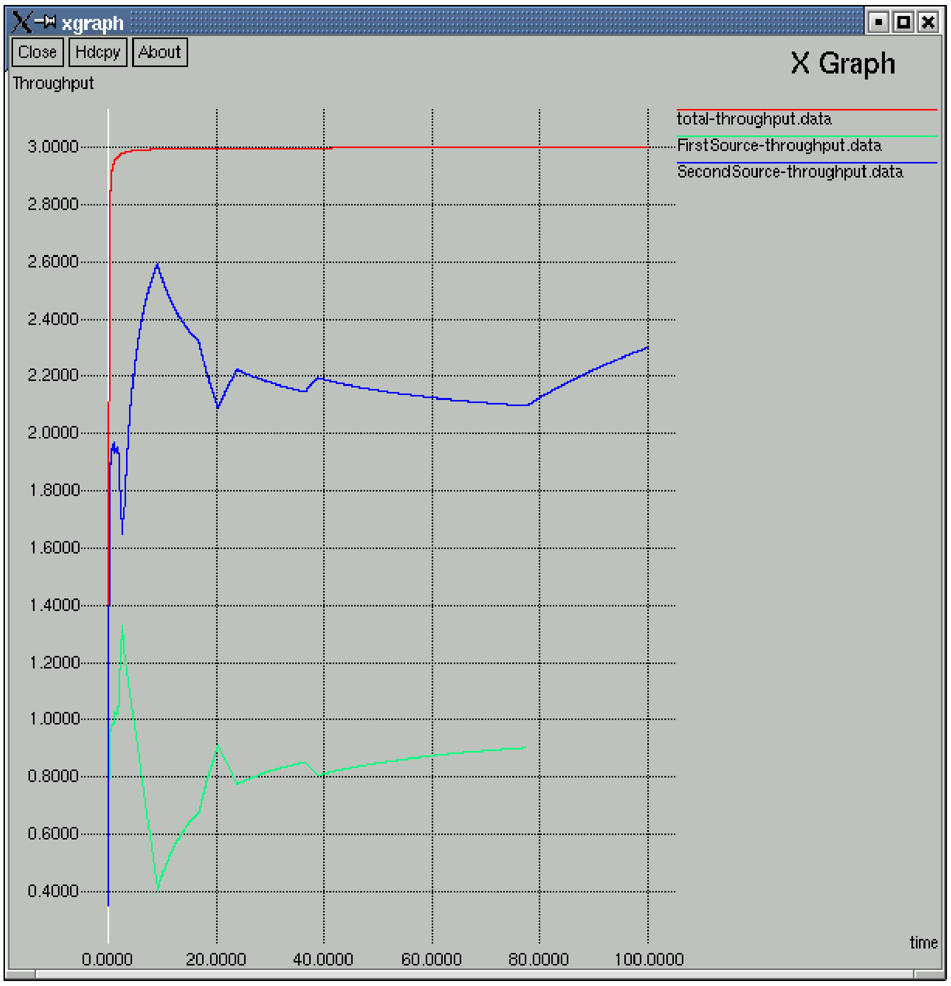


Figure 38: Scenario 2: Throughput

3.2.3 Scenario 3

We have used Scenario 2 with the addition of one extra source that sends traffic with a constant bit rate of 8 Mbit/sec to the same destination (see Figure 39). Also, the bottleneck link capacity has been increased to 12 Mbit/s. From Figure 40 it can be seen that there is a packet loss by all three sources with a fair way. The maximum throughput for all flows equals the bottleneck link capacity.

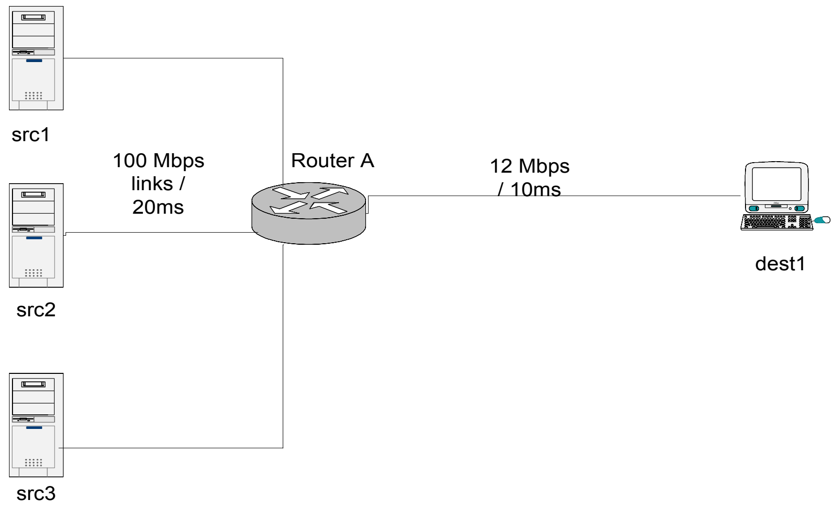


Figure 39. Scenario 3: Network topology

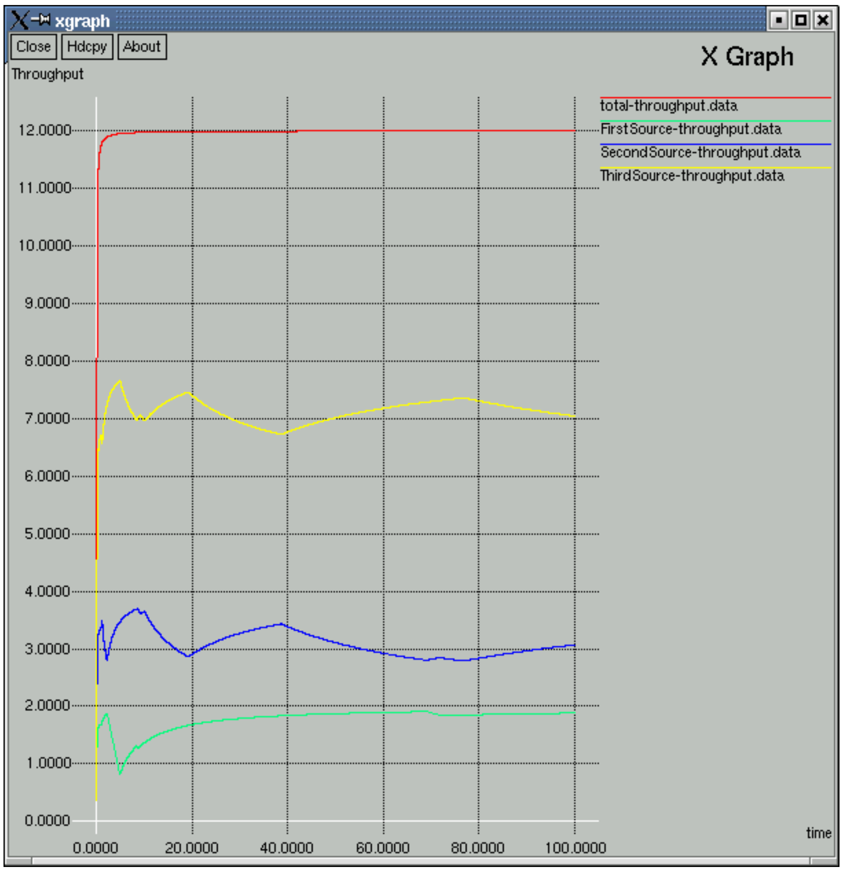


Figure 40: Scenario 3: Throughput

3.2.4 Scenario 4

The network topology for Scenario 4 is shown in Figure 41. Each of the four traffic sources initiates a TCP/FTP connection with a destination through a single router. This router uses RED as an active queue management mechanism. The bottleneck link capacity has been set to 10 Mbit/sec. We have also included a differentiation of the services offered. Specifically, the first two traffic sources have a higher priority than the last two traffic sources. From Figure 42, it can be seen that the packets sent by the first two traffic sources (belong to Assured class) have priority, by achieving a higher throughput than the other two traffic sources (belong to Best effort class).

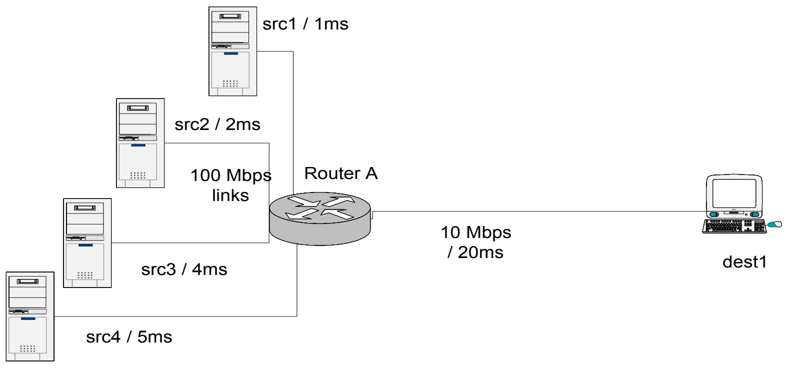


Figure 41: Scenario 4: Network topology

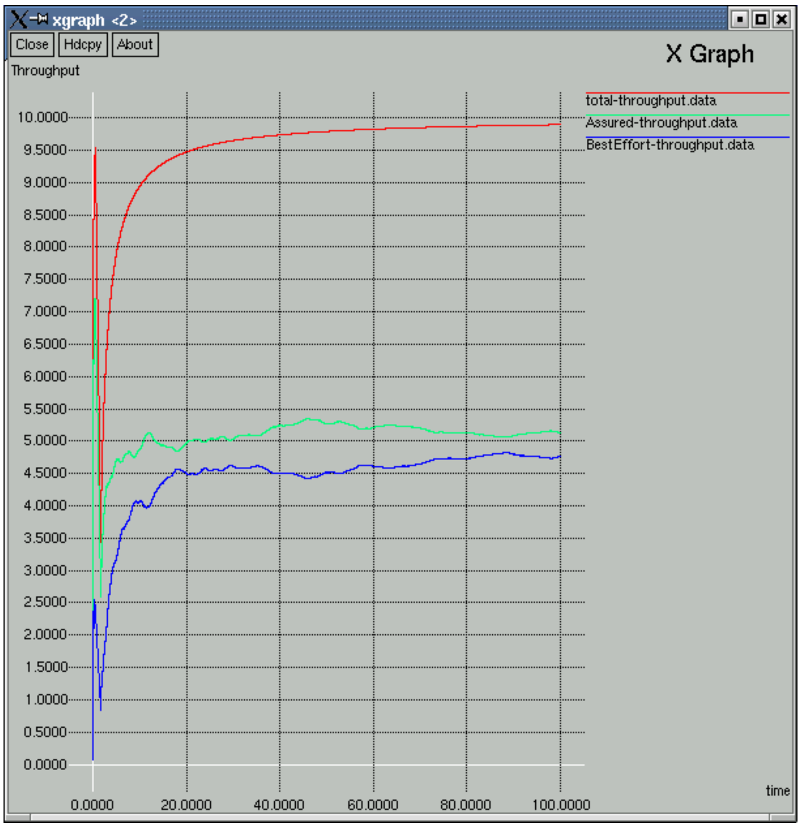


Figure 42: Scenario 4: Throughput

3.3 Simulative evaluation of existing IP architectures and protocols

The rapid growth of the Internet and increased demand to use the Internet for time-sensitive voice and video applications necessitate the design and utilization of effective congestion control algorithms. As a result, the differentiated services (Diff-Serv) architecture was proposed to deliver (aggregated) quality of service (QoS) in IP networks. Recently, many active queue management (AQM) schemes have been proposed to provide high network utilization with low loss and delay by regulating queues at the bottleneck links in TCP/IP networks, including random early detection (RED) [6], adaptive RED (A-RED) [7], proportional-integral (PI) controller [8], and random exponential marking (REM) [9]. Also, RIO [10] was proposed to preferentially drop packets. An AQM-enabled gateway can *mark* a packet either by dropping it or by setting a bit in the packet's header if the transport protocol is capable of reacting to explicit congestion notification (ECN). The use of ECN for notification of congestion to the end-nodes generally prevents unnecessary packet drops.

As part of ENDIKTIS project, we focus on the performance evaluation of the differentiated services for the provision of quality of service in IP networks. Particularly we investigate the provision of quality of service – that is high utilization, low loss and delay – by examining a number of representative queuing disciplines that provide congestion control. These schemes (mentioned above) are selected due to their availability in the simulation environment (NS-2).

3.3.1 Congestion control – Active queue management schemes of concern

Active queue management (AQM) mechanisms have recently been proposed, with the aim to provide high link utilization with low loss rate and queuing delay, while responding quickly to load changes. Several schemes have been proposed to provide congestion control in TCP/IP networks. RED [6], which was the first AQM algorithm proposed, simply sets some minimum and maximum *marking* thresholds in the router queues. The properties of RED have been extensively studied in the past few years, and many issues of concern have been arisen.

Recently, new proposed AQM mechanisms have appeared to give alternative solutions, and approached the problem of congestion control differently than RED.

Specifically, REM [9] algorithm uses the instantaneous queue size and its difference from a target value to calculate the *mark* probability based on an exponential law. Also, a PI controller [8] uses classical control theory techniques to design a feedback control law for the router AQM. It introduces a target queue length (TQL), in order to stabilize the router queue length around this value. Moreover, A-RED [7], proposed by the same author of RED [6], attempts to solve the problem for the need of tuning RED parameters. In particular, A-RED adjusts the value of the maximum *mark* probability to keep the average queue size within a target range half way between the minimum and maximum thresholds. Thus, A-RED maintains a desired average TQL twice the minimum threshold (if the maximum threshold is kept three times the minimum threshold). Furthermore, A-RED also specifies a procedure for automatically setting the RED parameter of queue weight as a function of the link capacity.

AQM mechanisms have also been proposed to preferentially drop non-contract conforming against conforming packets. The most popular algorithms used for such implementation are based on RED. The RED implementation for Diff-Serv, called RED In/Out (RIO) [10], defines that we have different thresholds for each class. Best-effort packets have the lowest minimum and maximum thresholds, and therefore they are *marked* earlier than packets of Assured class. They are also *marked* with a higher probability by setting the maximum *mark* probability higher than the one for packets of Assured class.

3.3.2 Simulation results

In this section we evaluate the performance and robustness of the existing AQM schemes, in a wide range of environments. We have taken some representative AQM schemes, namely A-RED [7], PI controller [8], REM [9], and RIO [10] using a recent version of NS-2 simulator (Version 2.1b9a). The simulation results are based on several scenarios-experiments that we have conducted.

The network topology used is shown in Figure 43. We use TCP/Newreno with an advertised window of 240 packets. The size of each packet is 1000 bytes. The buffer size of all queues is 500 packets. We use AQM in the queues of the bottleneck link

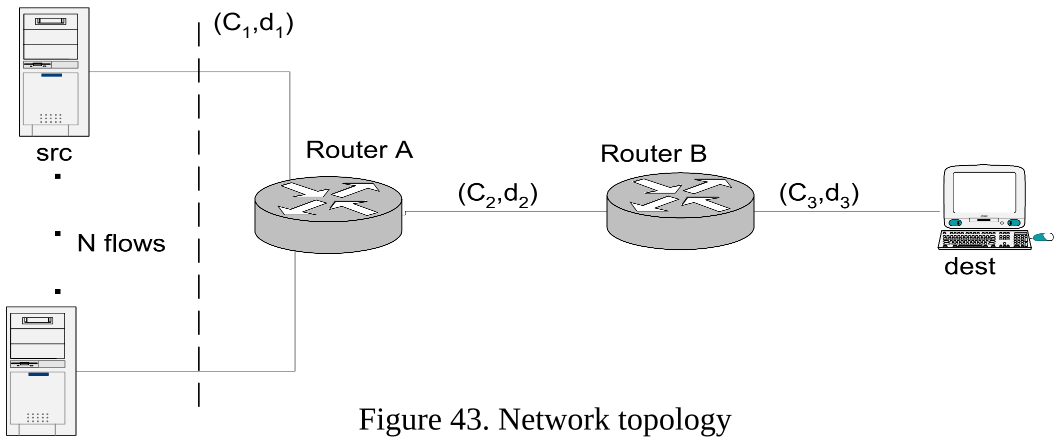


Figure 43. Network topology

between router-A and router-B. All other links have a simple Tail Drop (FIFO) queue. All sources (*N* flows) are greedy sustained FTP applications, except otherwise defined (where we also introduce web-like traffic). The links between all sources and router-A have the same capacity and propagation delay pair (C_1, d_1), whereas the pairs (C_2, d_2) and (C_3, d_3) define the parameters of the bottleneck link between router-A and router-B, and the link between router-B and the destination, respectively. The TQL of all AQM schemes, except otherwise defined, is set to 200 packets, as this is used in [8] (for A-RED, we set the minimum threshold to 100 packets, and the maximum to 300, giving an average TQL of 200 packets). The simulation time is *100 sec*.

The following experiments test the adequacy of existing mechanisms to provide quality of service in IP networks, that is. to investigate their ability to give high utilization, low losses and low queuing delays.

3.3.2.1 Scenario 1

In this scenario, we examine the ability of the AQM schemes to regulate the queue at the target value. The following parameter values are used: $N = 60$, (C_1, d_1) = (15Mbps, 40ms), (C_2, d_2) = (15Mbps, 5ms), and (C_3, d_3) = (30Mbps, 5ms), and TQL equals 200 packets. The results, shown in Figure 44, show that A-RED and REM shows good control performance, however, after a significant transient period with large overshoots, while PI controller spends considerably long time to regulate the queue to the reference value.

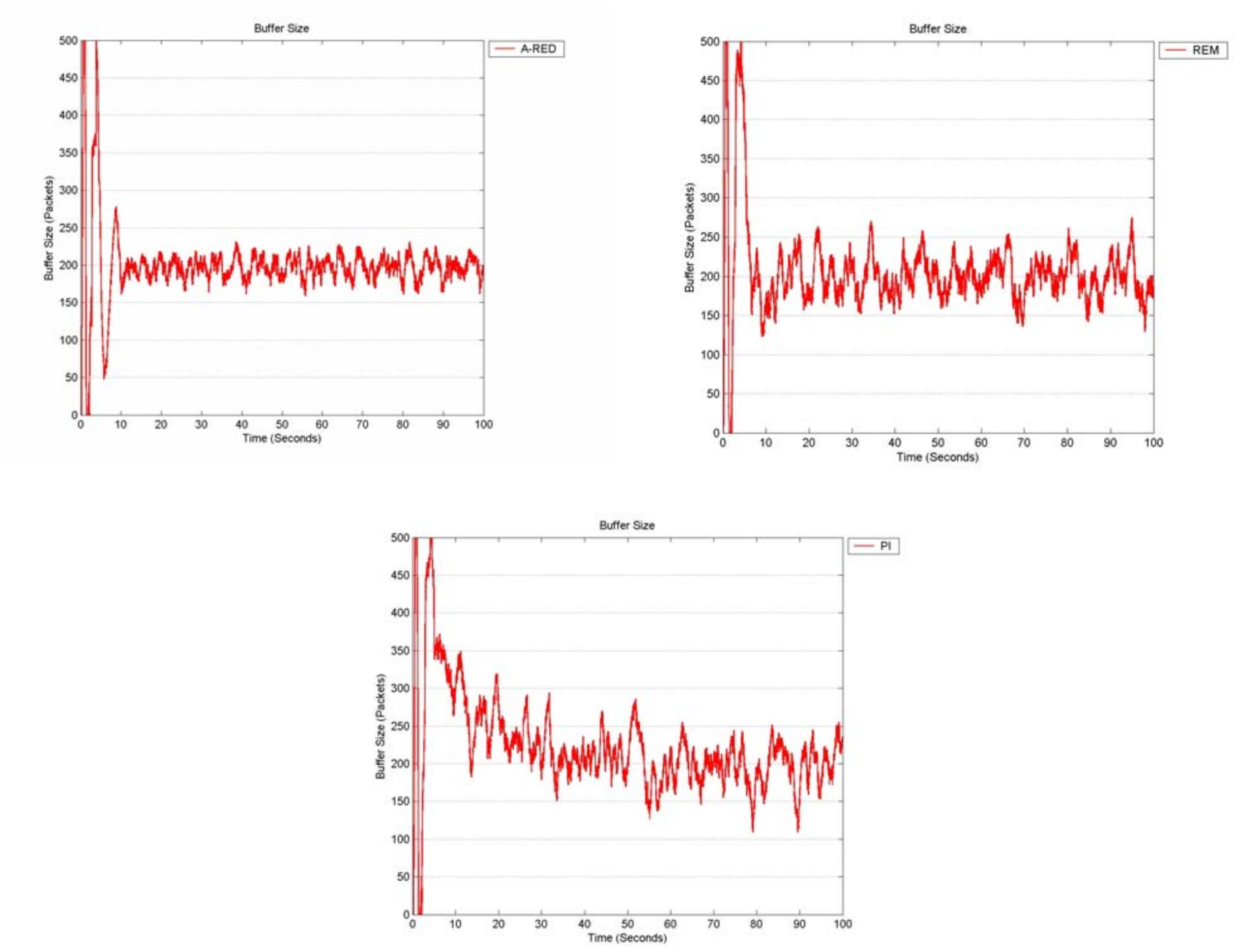


Figure 44: Scenario 1: Queue lengths.

3.3.2.2 Scenario 2

In order to explore the transient performance of the AQM schemes, we increase the number of flows from 60 to 100. The performance of the AQM schemes under dynamic traffic changes is also examined. We provide some time-varying dynamics by stopping half of the flows at time *t = 40 sec*, and resuming transmission at time *t = 70 sec*. The results (see Figure 45) show that PI and REM are not as robust against the dynamic traffic changes (especially in the case of PI), as they are slow to settle down to the reference value, resulting in large queue fluctuation. A-RED responds well, except for some large overshoots at the time of the traffic changes.

3.3.2.3 Scenario 3

In this scenario, we investigate the performance of AQM schemes under higher link capacities and propagation delays, that is, we set (C_1, d_1) = (100Mbps, 5ms), (C_2, d_2)

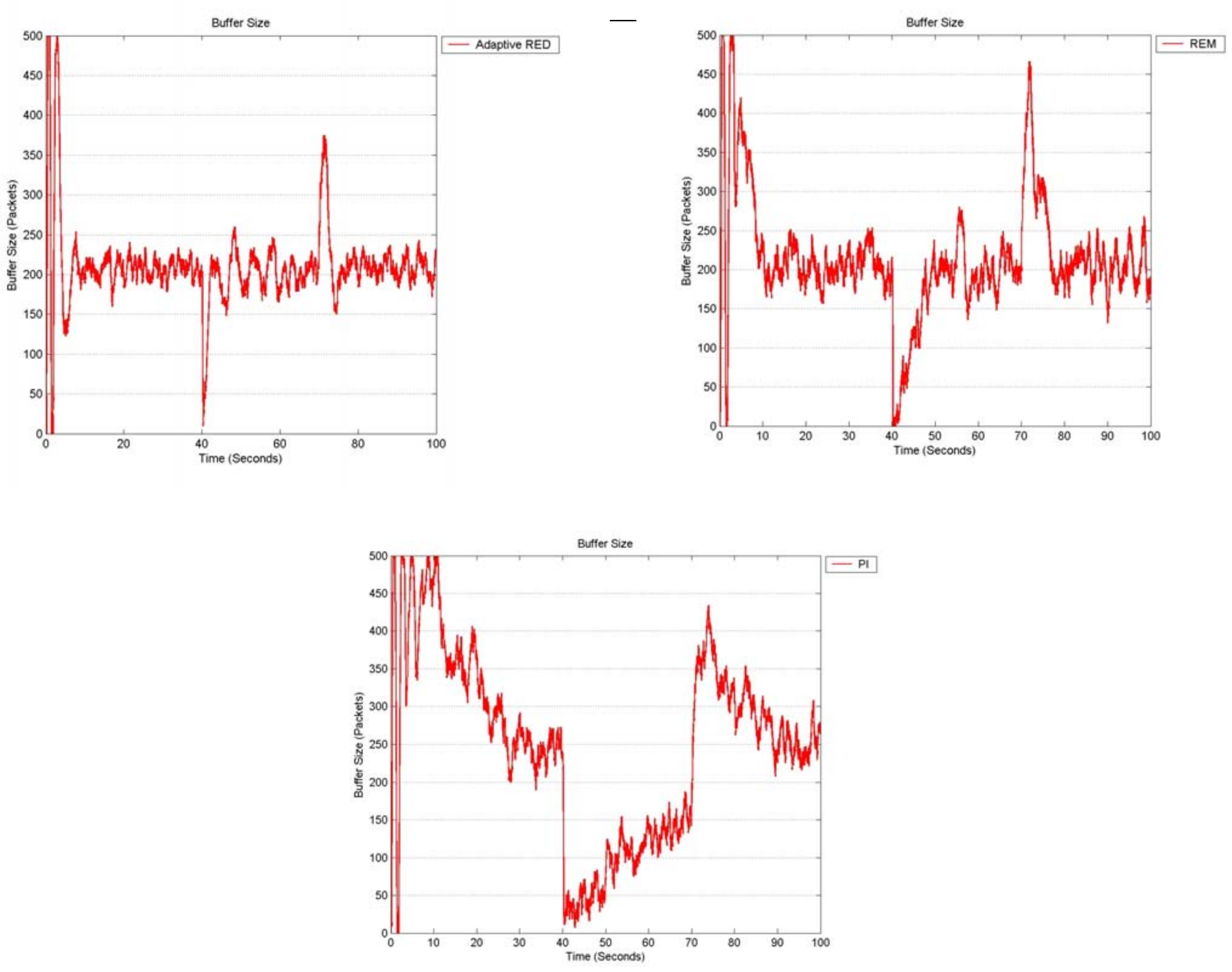


Figure 45: Scenario 2: Queue lengths.

= (15Mbps, 120ms), and (C_3, d_3) = (200Mbps, 5ms), while $N = 100$. We also keep the time-varying dynamics on the network, as used in *Scenario 2*. We specifically examine the effect of the round-trip time (RTT) by increasing the propagation delay of the bottleneck link (i.e., *120 ms*). In general, an increase of RTT degrades the performance of an AQM scheme. The results (see Figure 46) show that PI, A-RED, and REM exhibit large queue fluctuations that result in degraded utilization and high variance of queuing delay.

3.3.2.4 Scenario 4

We also investigate the effect of the traffic load factor (N) in the last experiment, by increasing N from 100 to 200, 300, 400, and 500. The expected queuing delay experienced at router-A is 106.7 ms (15Mbps link capacity corresponds to 1875 packets/sec; for a TQL of 200 packets the expected mean delay is 200/1875 = 0.1067. Note that the parameters of bottleneck link capacity and TQL are the same as in [9]).

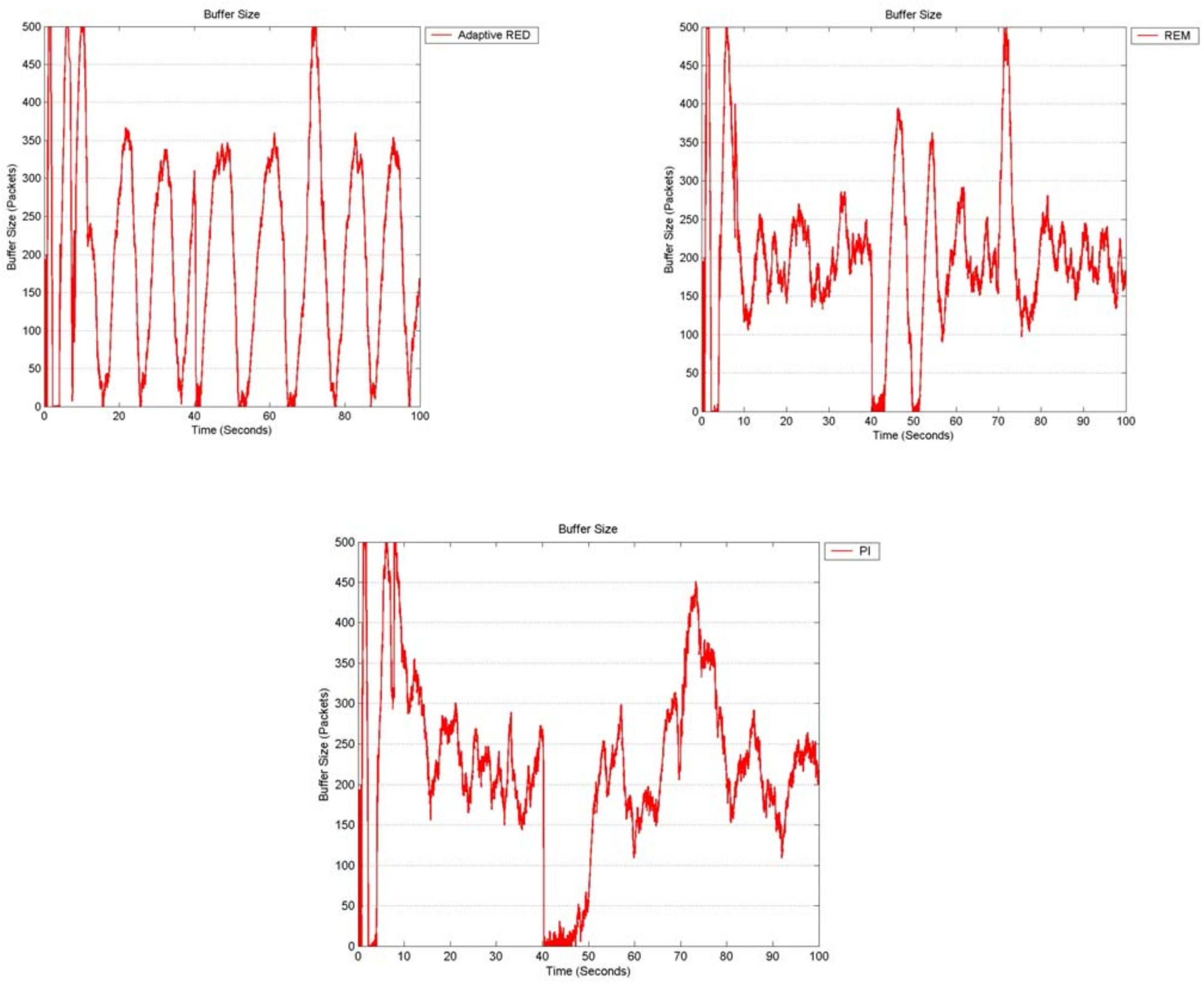


Figure 46: Scenario 3: Queue lengths.

Figure 47 shows the loss rate as traffic load increases, where it can be seen that A-RED has the largest drops with a large increase of packet loss with respect to higher loads. Figure 48 shows the utilization of the bottleneck link with respect to the mean queuing delay, where the AQM schemes show a poor performance as the number of traffic load increases, achieving low link utilization, and large queuing delays, far beyond the expected value. Table 51 lists the statistical results of the mean queuing delay and its standard deviation. It is clear that the AQM schemes exhibit very large queue fluctuations with large amplitude that inevitably deteriorates delay jitter.

We have further conducted the same experiment, by setting the bottleneck link propagation delay to *60 msec*. Figure 49 shows the loss rate as traffic load increases, and Figure 50 shows the utilization of the bottleneck link with respect to the mean queuing delay. As it can be observed, similar results are obtained as above.

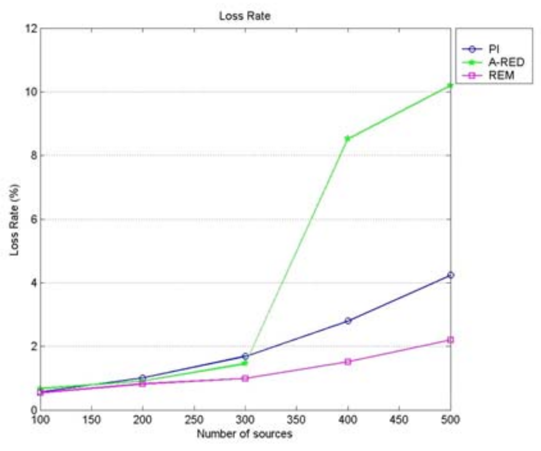


Figure 47: Scenario 4.
(prop. delay = 120 msec):
Loss Rate vs Traffic Load
(for 100- 500 flows)

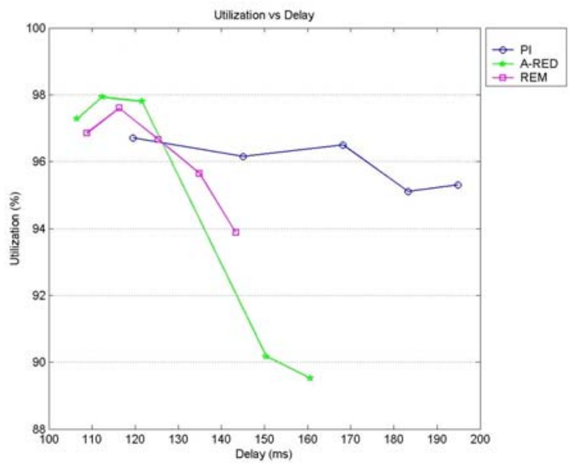


Figure 48: Scenario 4.
(prop. delay = 120 msec):
Utilization vs Mean Delay
(for 100- 500 flows)

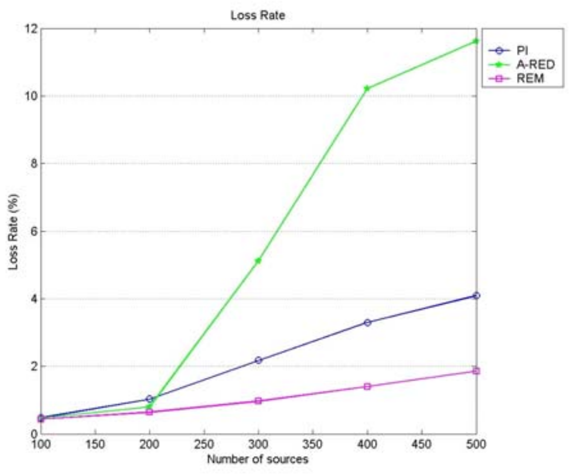


Figure 49: Scenario 4.
(prop. delay = 60 msec):
Loss Rate vs Traffic Load
(for 100- 500 flows)

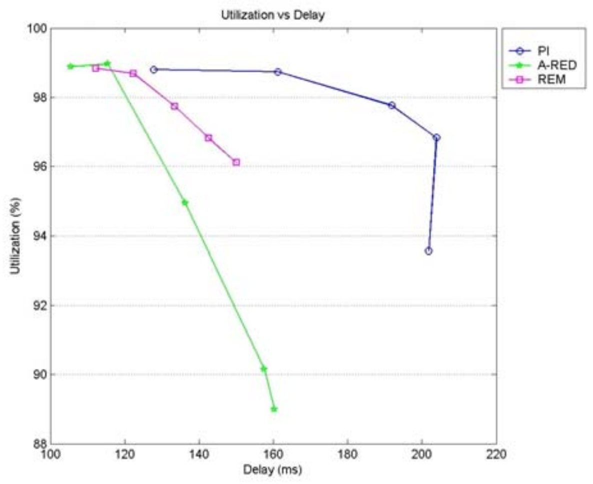


Figure 50: Scenario 4.
(prop. delay = 60 msec):
Utilization vs Mean Delay
(for 100- 500 flows)

3.3.2.5 Scenario 5

We further investigate the performance of AQM schemes by introducing additional web-like traffic that can be seen as noise-disturbance to the network. In particular, we keep the same parameters as in *Scenario 4*, without the time-varying dynamics. The number of flows is kept to 100 for FTP applications, with an additional 100 web-like traffic flows. We have conducted experiments for two specific values of the TQL (i.e., 100 and 200 packets) to examine the robustness of the AQM schemes. For both cases the results are shown in Table 52 where we obtain the mean queuing delay and its standard deviation, link utilization and loss rates. It is clear that, for both cases, the

AQM schemes exhibit very large variations of the queue; consequently, this has the effect of having degraded link utilization with large number of drops.

Traffic Load	AQM schemes	Mean-Delay (ms)	Std-Deviation (ms)
100 Sources	PI	119.508	54.8057
	ARED	106.531	72.8443
	REM	108.769	47.7956
200 Sources	PI	144.998	85.6514
	ARED	112.356	52.2939
	REM	116.298	50.1747
300 Sources	PI	168.225	96.2637
	ARED	121.653	51.1104
	REM	125.403	63.0991
400 Sources	PI	183.278	99.527
	ARED	150.439	66.1591
	REM	134.916	75.5712
500 Sources	PI	194.903	94.0823
	ARED	160.633	58.7155
	REM	143.333	82.2324

Table 51. Scenario 4: Summary of mean delay and standard deviation

Target Queue Length	AQM schemes	Mean-Delay (ms)	Std-Deviation (ms)	Utilization (%)	Loss rate (%)
TQL 100 (expected mean delay: 53.3 ms)	PI	69.6015	44.9733	97.9	0.56
	ARED	57.2572	42.6883	97.6	0.61
	REM	57.5126	32.8804	97.9	0.49
TQL 200 (expected mean delay: 106.7 ms)	PI	136.754	37.9652	97.92	0.65
	ARED	108.91	69.9759	97.5	0.63
	REM	108.629	32.6228	97.89	0.52

Table 52. Scenario 5: Summary of statistical results

3.3.2.6 Scenarios 6-10

We further investigate the performance of IP networks under preferential packet control. We consider two different traffic classes: Assured traffic class, which has the highest priority, and best-effort traffic class, which has the lowest priority in a buffer queue. The most popular algorithm used in such cases is based on RED, namely RED In/Out (RIO). As RIO is already integrated in NS-2 simulation environment, we examine RIO capabilities to provide QoS. For RIO, the minimum and maximum thresholds, for best-effort traffic, are set to 50 and 150 packets, respectively. The equivalent values for assured traffic are 100 and 300 packets, respectively. The maximum *mark* probability for best-effort traffic is set to 0.1, whereas the one for assured traffic is set to 0.02.

Scenarios 6-10 use the network topology shown in Figure 43, with TCP/FTP traffic. In these scenarios, we use TCP/Newreno with an advertised window of 240 packets. The size of each packet is 1000 bytes. The buffer size of all queues is 500 packets. We use AQM in the queues of the bottleneck link between router-A and router-B. The link capacities and propagation delays are set as follows: (C_1, d_1) = (100Mbps, 5ms), (C_2, d_2) = (15Mbps, 120ms), and (C_3, d_3) = (200Mbps, 5ms), while $N = 100$.

All results are summarized in Table 53, where the performance-QoS metrics are the bottleneck link utilization, the loss rate and the mean queuing delay with its standard deviation.

Scenario 6 considers a limited number of flows tagged as assured class traffic; 2 out of 100 flows are considered belonging to assured class, whereas the rest, 98 flows, are tagged as best-effort. Figure 51, shows the queue of RIO, where we can observe that RIO exhibits very large queue fluctuations that results in degraded utilization, losses and high variance of queuing delay (see Table 53). Furthermore, RIO cannot provide sufficient link utilization for assured class traffic.

Scenario 7 increases the number of flows tagged as assured traffic class to 10. RIO slowly regulates its queue (see Figure 52), after a significant transient period with large overshoots that results in degraded utilization and significant amount of losses. Furthermore, RIO fails to provide adequate discrimination between the two traffic classes.

Scenario 8 examines the behavior of RIO under dynamic traffic changes. We use the previous experiment, and provide some time-varying dynamics by stopping the assured-tagged flows at time $t = 40$ *sec*, and resuming transmission at time $t = 70$ *sec*.

The results (see Figure 53) show that RIO is not robust against the dynamic traffic changes.

Scenario 9 increases the number of flows tagged as assured traffic to 90. In the presence of large amount of assured traffic, compared with the best-effort traffic RIO exhibits large queue fluctuations that result in high losses (see Figure 54).

Scenario 10 uses the previous experiment, and examines the effect of the RTT by having heterogeneous propagation delays of the links between the sources and router-A (we separate the 100 flows into groups of 10, and for each group - that consists of 9 assured-tagged flows and 1 best-effort-tagged flow – its propagation delay is increased by *5 msec*, starting from *5 msec* up to *50 msec*). The propagation delay of the bottleneck link has also changed to *60 msec*. The results (see Figure 55) show that RIO exhibits large queue fluctuations, worst than the previous experiment, that result in a significant amount of losses and high variance of queuing delay.

Scenarios	**AQM**	**Utilization (%)**			**Loss Rate (%)**			**Delay (ms)**	
		Best-effort	**Assured**	**Total**	**Best-effort**	**Assured**	**Total**	**Mean-Delay**	**Std-Deviation**
6	**RIO**	94.33	2.27	96.6	1.66	1.72	1.67	178.52	79.72
7	**RIO**	44.67	50.8	95.47	5.77	0.22	2.97	112.86	55.97
8	**RIO**	65.6	25.4	91	3.7	0.435	2.84	87.17	79.7
9	**RIO**	0.6	96.47	97.07	15.12	1.12	1.22	158.8	46.22
10	**RIO**	0.26	97.4	97.66	30.5	2.09	2.2	155.70	60.91

Table 53. Scenarios 6-10: Summary of statistical results

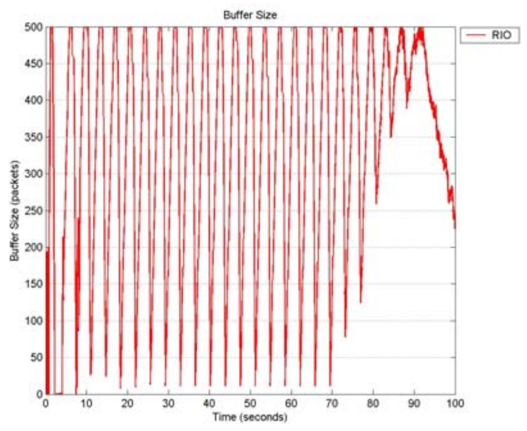


Figure 51. Scenario 6: Queue length

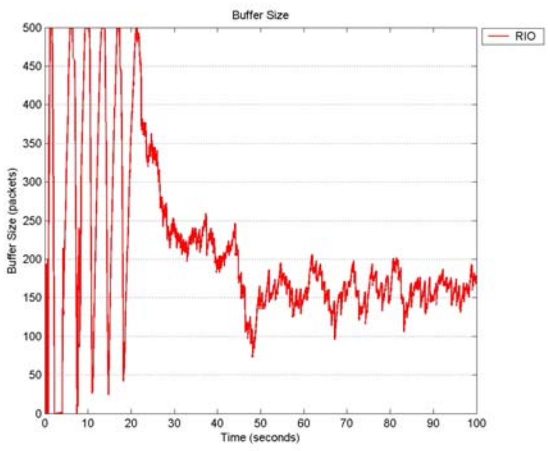


Figure 52. Scenario 7: Queue length

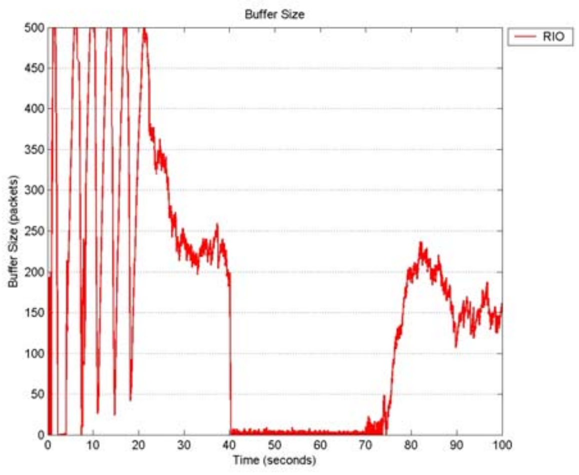


Figure 53. Scenario 8: Queue length

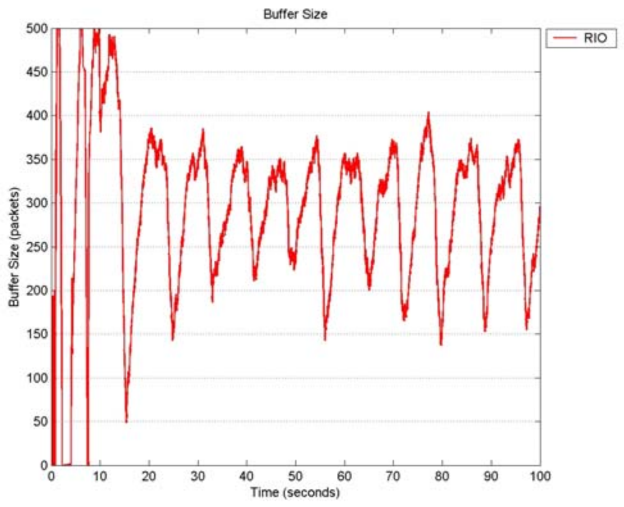


Figure 54. Scenario 9: Queue length

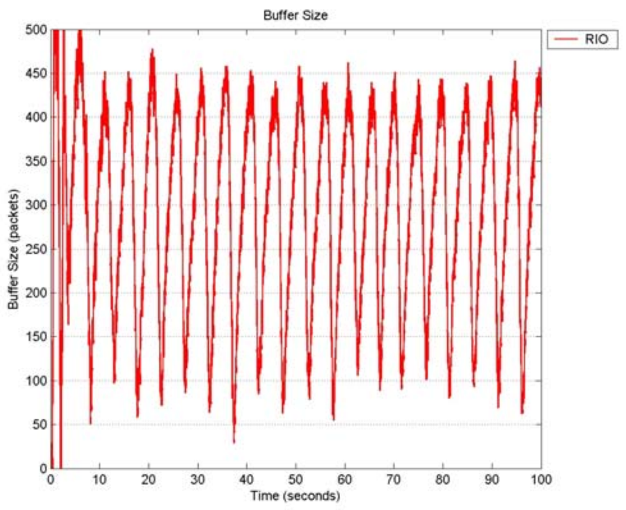


Figure 55. Scenario 10: Queue length

4 Conclusions

This deliverable presented extensive experimental and simulative results by evaluating IP architectures and protocols of concern. Several representative scenarios and measurements were made with the aid of both simulation environments and pilot networks.

We have mainly concentrated on the differentiated services for the provision of quality of service in IP networks. In particular the behavior and performance of existing congestion control and queuing disciplines are evaluated in order to examine the ability of such mechanisms to provide adequate quality of service. The most critical characteristics of quality of service – identified by Deliverable 1 - such as throughput capacity/utilization, losses and delay variations, are considered.

The results of the experiments and simulations show that the existing mechanisms in today’s Internet are not as robust and effective, in cases of dynamic network/traffic changes. Therefore, there is a need for further investigation, improvement and development of new mechanisms that can provide effective and efficient quality of service.

5 References

[1] W. Almesberger, Linux Network Traffic Control- Implementation Overview, Technical Report EPFL ICA, April 1999.
[2] W. Almesberger, J. Hadi Salim, and A. Kuznetsov. Differentiated services on Linux. Internet draft, draft-almesberger-wajhak-diffserv-linux-00.txt, work in progress, February 1999.
[3] S. Radhakrishman, Linux- Advanced Networking Overview, V1, Department of Electrical Engineering & Computer Science, University of Kansas.
[4] OPNET Modeler, Homepage, http://www.opnet.com/products/modeler.
[5] Network Simulator, NS-2, Homepage, http://www.isi.edu/nsnam/ns/.
[6] S. Floyd, V. Jacobson, "Random Early Detection gateways for congestion avoidance", IEEE/ACM Trans. on Networking, Aug. 1993.
[7] S. Floyd, R. Gummadi, S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management", Technical report, ICSI, August 2001.
[8] C. V. Hollot, V. Misra, D. Towsley, W.-B. Gong, "Analysis and Design of Controllers for AQM Routers Supporting TCP Flows" IEEE Transactions on Automatic Control, vol. 47, no. 6, pp. 945-959, June 2002.
[9] S. Athuraliya, V. H. Li, S. H. Low, Q. Yin, "REM: Active Queue Management", IEEE Network Magazine, 15(3), pp. 48-53, May 2001.
[10] D. Clark, W. Fang "Explicit Allocation of Best Effort Packet Delivery Service", IEEE/ACM Transactions on Networking, Vol. 6, No. 4, pp. 362-373, August 1998.